DNN model extraction attacks using prediction interfaces

Alexey Dmitrenko

School of Science

Thesis submitted for examination for the degree of Master of Science in Security and Mobile Computing.
Espoo June 26, 2018

Supervisors

Prof. N. Asokan, Aalto University
Prof. Danilo Gligoroski, NTNU

Advisors

M.Sc. (Tech) Mika Juuti

PhD Samuel Marchal

Aalto University
School of Science

Aalto University
School of Science

Aalto University, P.O. BOX 11000, 00076 AALTO
www.aalto.fi
Abstract of the master's thesis

Author Alexey Dmitrenko

Title DNN model extraction attacks using prediction interfaces

Degree programme Security and Mobile Computing

Major Security and Mobile Computing **Code of major** T3011

Supervisors Prof. N. Asokan, Aalto University
Prof. Danilo Gligoroski, NTNU

Advisors M.Sc. (Tech) Mika Juuti, PhD Samuel Marchal

Date June 26, 2018 **Number of pages** 68 **Language** English

Abstract

Machine learning (ML) and deep learning methods have become common and publicly available, while ML security to date struggles to cope with rising threats. One rising threat is model extraction attacks where adversaries are able to reproduce a target model close to perfection. The attack is widely deployable since the attacker needs only to have access to predictions to perform this attack. Stolen ML models could either be used for personal advantage to abuse paid prediction services or to create transferable adversarial examples that can be used to undermine the integrity of prediction services, i.e. prediction quality. This is a significant threat in several application areas, such as in autonomous driving, which rely heavily of computer vision via deep neural networks.

In this thesis, we reproduce existing model extraction attacks and evaluate novel techniques to extract deep neural network (DNN) classifiers. We introduce new synthetic query generation strategies, and demonstrate their efficiency at extracting models for creating transferable targeted adversarial examples from stolen DNNs.

Keywords MLaaS, DNN extraction , Adversarial ML, Transferability

Preface

The thesis work was done by the author under the guidance of Professor N. Asokan of Aalto University, Professor Danilo Gligoroski of NTNU and my advisors at Aalto University M.Sc. (Tech) Mika Juuti and PhD Samuel Marchal. I would like to express my sincerest gratitude and appreciation to Mr. Mika Juuti, Mr. Samuel Marchal and Professor N. Asokan for their extended guidance, valuable comments and feedback throughout the thesis.

Otaniemi, 26.06.2018

Alexey Dmitrenko

Contents

Abbreviations

API Application Programming Interface

CNN Convolutional Neural Network

DNN Deep Neural Network

FGSM Fast Gradient Sign Method

GPU Graphics Processing Unit

ML Machine Learning

MLaaS Machine-Learning-as-a-Service

NCS Neural Compute Stick

PGD Projected Gradient Descent

ReLU Rectified Linear Unit

VPU Vision Processing Unit

XaaS Everything-as-a-Service

1 Introduction

1.1 Motivation

Machine learning (ML) is increasingly being deployed by many companies and start-ups as part of the everything-as-a-service (XaaS) paradigm [3, 32, 14]. Machine Learning-as-a-Service (MLaaS) is a new service concept that pushes machine learning model prediction, and sometimes training too, into the cloud. The market slice is expected to grow rapidly in the coming five years. For example, according to the MarketsandMarkets^{TM} the MLaaS market is estimated to grow from USD 613.4 Million in 2016 to USD 3,755.0 Million by 2021 [29]. This increase is expected to happen due to a change in availability and price of ML services. Before, only a few companies had enough data and know-how to use Machine Learning for large datasets. With the introduction of MLaaS, it is possible for everyone to use such services and without hardware concerns. For providers, it is also beneficial because it becomes a new business field that companies can monetize.

Recent changes in data protection regulation (i.e General Data Protection Regulation, GDPR) [9] introduced several difficulties for ML. GDPR requires all decision made by machine algorithm to be clearly explained ("right to explanation" [6]) making it difficult to deploy deep learning techniques due to their complexity and non-linear decisions. Exposure of certain models to data leakage and theft are serious threats. For example, banks and other financial institutes that use ML methods to decide whether a person is eligible for a mortgage now should explain why a particular decision is made. Therefore, a clever adversary can exploit such fact and try to use this information from numerous applications to infer decision boundaries of the financial model and gain financial advantage of it.

There are several other problems in ML that are related to information security. In case of a health-care system, for example, models process patient data and it should be kept private. However, there are ways to disclose such information if no security measures are applied to algorithms [41, 7]. Other security vulnerabilities in machine learning are related to SPAM filters, network security software, security information event management systems and other types of classifier systems. If an adversary manages to obtain information about the model that is used in any of these applications, then he may try to execute an evasion attack, to evade detection by such a system. In the evasion attack scenario, a malicious user typically tries to add

adversarial perturbation to natural samples which are misclassified by the model [36, 35]. Such samples can violate service usability or model integrity (prediction quality). Evasion attacks are not the only threat in exposing ML models. There are also inversion attacks where the goal of an adversary is to obtain a part of a training set via analysis of a ML model.

Undoubtedly, all these issues can happen whether systems are deployed using company resources in a company network, or if they are in the cloud infrastructure on an untrusted third-party server. However, threats against ML systems apply not only when in the cloud settings, but in locally available models as well. In fact, the threat surface increases when the attacker has access to the model, meaning that the attacker can do more powerful attacks. For instance, with the huge popularity of IoT devices and smart infrastructures, ML models are widely deployed inside hardware processing units. For example, Vision Processing Unit (VPU) from Intel Movidius is extensively used in security cameras[1]. These hardware chips are designed to accelerate deployability of machine intelligence by having low power consumption. Because of concerns about misuse of the models they are expected to be hardware-protected by a Trusted Execution Environment e.g. Intel SGX [39]. Be it as it may, prediction APIs will always be queryable in such models for practical reasons. As we discuss in this work, an adversary only needs access to the prediction API in order to perform model extraction attacks. In this thesis, we analyze the applicability of model extraction attacks on DNNs and propose novel attacks. The main goal is to understand the risk of model extraction attacks for service providers and its applicability in real scenarios.

1.2 Contribution

In this thesis we claim the following contributions:

- We reproduce previous state-of-the-art attack methods [41, 36]. We evaluate the attack effectiveness of previous techniques and isolate several variables that contribute to the effectiveness of model extraction (Section 6). We analyze the effect of each separately. We validate their performance on both previously reported and additional dataset [36] (Section 2.6).
- We propose novel query strategies along with the algorithm to extract DNN

[1] https://developer.movidius.com/

models using only prediction APIs. The method extends previous work [36] (Section 4).

- We test and evaluate proposed techniques to steal DNNs on a neural compute device (embedded system). We show that extraction of local models is feasible (Section 6).

1.3 Structure

The remainder of the work is structured as follows: Section 2 provides with basic descriptions of algorithms and definitions that are used throughout this work. We give an overview of previous attacks that used as a baseline in the evaluation. Section 3 presents the problems discussed in this thesis, identifies the threat model and lists requirements for new DNN extraction techniques. Section 4 introduces a generic approach to extract a ML model and describes our new query strategy to improve existing attacks. Section 5 presents datasets used for evaluation of the attack along with different learning strategies for the attack algorithm that we propose will have a positive impact on the extraction of DNN models. Section 5.3 gives an overview of experimental setups that we use to evaluate our work and describes DNN architectures we used. Section 6 gives an overview of performance metrics that are used to evaluate previous attacks as well as new methods, shows the evaluation of initial samples impact, synthetic query impact, training strategy impact and the effect of varying training time of proposed solution and previous attacks. Section 7 discusses the related work to DNN model extraction and defenses to date. We conclude this thesis by presenting conclusions and describing possible future work in Section 8.

2 Background

In this section, we address all the necessary definitions of terms, techniques that are closely related to the thesis. We present definitions of important concepts in order to give a better understanding of the solutions and evaluations described in this work.

2.1 Mathematical background

2.1.1 Convolution

Convolution is a mathematical operation applied on two functions f and g that produces a third function $f * g$. Usually, the result of the convolution considered to be an altered version of f or g. The convolution operation can be interpreted as the "similarity" of one function to the reflected and shifted copy of another. In other words, assume $f, g : \mathbb{R}^d \to \mathbb{R}$ are integrable functions in $\mathbb{R}^d$. Therefore, the convolution of those functions is $f * g : \mathbb{R}^d \to \mathbb{R}$ and it can be described as the following mathematical expression:

$$(f * g)(x) = \int_{\mathbb{R}^d} f(y) \times g(x - y)dy = \int_{\mathbb{R}^d} f(x - y) \times g(y)dy \tag{1}$$

2.1.2 Gradient

Gradient ∇ is a vector that indicates the direction of the greatest increase of a certain value ϕ. The absolute value of a gradient $\nabla\phi$ is equal to the rate of growth of this value in this direction. It can be described with following mathematical expression:

$$\nabla\phi = \frac{\partial\phi}{\partial x} \times \mathbf{i} + \frac{\partial\phi}{\partial y} \times \mathbf{j} + \frac{\partial\phi}{\partial z} \times \mathbf{k} \tag{2}$$

Where $\mathbf{i}, \mathbf{j}$ and $\mathbf{k}$ are unit vectors in the directions of x, y and z axis correspondingly.

2.1.3 Chain rule

Chain rule allows computing the derivative of the composition of two or more functions. Consider function f is differentiable at a point x_0 and a function g is differentiable at a point $y_0 = f(x_0)$, thus their composition $h = f \circ g$ is also differentiable at that point and its derivative is equal to

$$h'(x_0) = g'\Big(f(x_0)\Big) \cdot f'(x_0) \tag{3}$$

2.1.4 Jacobian matrix

Jacobian matrix is a matrix that consists of gradient values of a function $\mathbf{f}$ [8]. If a function $\mathbf{f}$ is differentiable at a point x and have all partial derivatives at x then there is a Jacobian matrix J that shows linear approximation of $\mathbf{f}$ near the point x. It can be described as the following mathematical expression:

$$\mathbf{J} = \frac{d\mathbf{f}}{d\mathbf{x}} = \begin{bmatrix} \frac{\partial \mathbf{f}}{\partial x_1} \cdots \frac{\partial \mathbf{f}}{\partial x_n} \end{bmatrix} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \cdots & \frac{\partial f_m}{\partial x_n} \end{bmatrix} \tag{4}$$

2.1.5 p-norm

A norm is a function that is defined on a vector space and generalizes the definition of vector length and absolute value. The norm of a vector V over a field of real ($\mathbb{R}$) numbers can be described as a function $p : V \to \mathbb{R}$ that has following properties:

$$\begin{aligned} &1.\ p(x) = 0 \implies x = 0_V; \\ &2.\ \forall x, y \in V, p(x + y) \leq p(x) + p(y); \\ &3.\ \forall \alpha \in \mathbb{R}, \forall x \in V, p(\alpha x) = |\alpha| p(x). \end{aligned} \tag{5}$$

These properties are called the norm's postulates [1]. Usually, norms are written as $\| \cdot \|_p$ and defined as:

$$\|v\|_p = \left(\sum_{i=1}^{n} |v_i|^p \right)^{\frac{1}{p}} \tag{6}$$

The most commonly used norm in $\mathbb{R}$ space is the Euclidean norm or L_2 norm. It has the following expression: $\|X\|_2 = \sqrt{\left(x_1^2 + \cdots + x_n^2\right)}$, where n is the dimensionality of X. It is commonly seen as an expression describing the length of a vector.

2.1.6 L^p space

L^p space is a mathematical function space that is defined using p-norm in finite dimensions [1]. Often it is used as a distance metric between two objects in L^p space. The most common distance metrics using L_p norm in machine learning are L_0, L_1, L_2 and L_∞ General formulas for L_p distances can be written as $\|x - x'\|_p$.

- L_0 distance for vector x and x' assesses the number of elements that differ. For example, in case of images, it shows the total number of pixels which are different in one image comparing to the other

- L_1 distance for vector x and x' assesses the summed absolute value discrepancy between x and x'. In case of images, it is the sum over all pixels absolute value discrepancies.

- L_2 distance for vector x and x' assesses Euclidean distance between two objects. It is the most common distance metric in small-dimensional space ($n \leq 3$), but it loses its practical usefulness in high-dimensional space ($n \geq 3$). It was shown in [2] that as the n increases the Euclidean distance between a point and its closest neighbor, and between that point and its furthest neighbor changes in non-obvious way affecting the results.

- L_∞ for vector x and x' assesses the maximum change in any dimension. $\|x - x'\|_\infty = \max(|x_1 - x'_1|, ..., |x_n - x'_n|)$, where x_i denotes x's value in dimension i. In images it means that the discrepancy value between two images is always limited to some maximum value, but at the same time, any number of pixels may be modified.

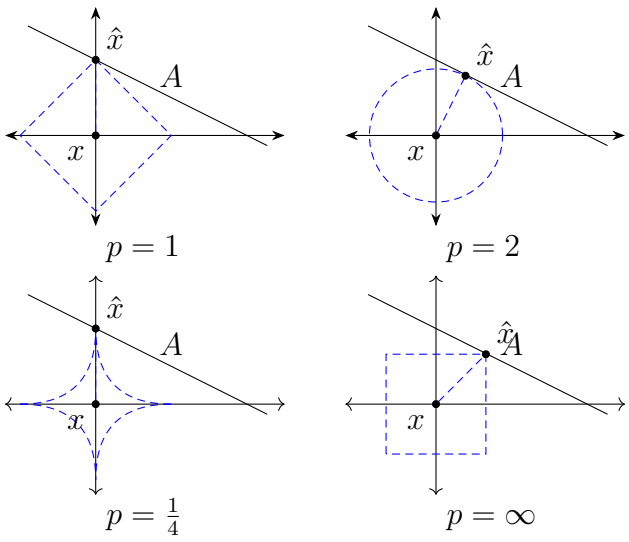

Figure 1: P-space graphical representation with unit circles.

Graphical representation of L^p space for p *in* $[1, 2, \frac{1}{4}, \infty]$ using unit circles can be found in Figure 1. The dashed shapes denote equal distance from origin in different L^p spaces. The case when $0 \leq p < 1$ is different and Equation 6 is not applicable. The vector space in this case is not locally convex and the same formula cannot be applied. There is a distance metric instead that defines the case when $0 \leq p < 1$,

that is

$$d_p(x, y) = \sum_{i=1}^{n} |x_i - y_i|^p \tag{7}$$

Therefore, we show $p = \frac{1}{4}$ as an example for simplicity. Lower values of p converge to axis lines and are hard to represent on the graph.

2.2 Machine Learning

ML is a subclass of artificial intelligence methods. The main characteristic of ML is not to provide a direct solution to a particular problem, but to do so-called learning while solving a great number of similar problems. Training occurs with the use of mathematical statistics, numerical methods, optimization methods, probability theory, graph theory, and various techniques of working with data in digital form. Training is commonly divided into two types:

1. Supervised learning or learning from examples, is a type of training based on inductive learning and identification of empirical regularities in the data. Many methods are closely related to information extraction and data mining techniques.

2. Unsupervised learning (self-learning, spontaneous learning) is the type of training describing internal interrelations, dependencies and regularities existing between objects.

Unsupervised learning is often contrasted with supervised learning, where each learning object is forced to have a "correct answer" and there is a need to find a relationship between the system's stimuli and responses.

Classification, the area of predicting categorized labels of samples, is the most common form of supervised learning. A great deal of real-world problems can be cast as classification problems. For instance, classifying digits and traffic signs are fundamentally the same problem, and can be solved with similar algorithms. We will focus on supervised techniques in this thesis.

2.2.1 Training ML model

In order to train a ML model one need to find optimal parameters θ that minimize a classification error rate. Usually, the error rate described by cost function C. Overall, the goal of training is to minimize a function C by finding an optimal point θ that

solves $\frac{\partial C(\theta)}{\theta} = 0$. However, it is mathematically infeasible to find the solutions of this equation so instead numerical optimization methods are applied during training. The most common supervised learning optimization method to train a ML model is a gradient descent. **Gradient descent** is an iterative method of finding local minimum of a function C using gradient of that function. The method takes steps towards the negative gradient of a function at a point θ. One step can be viewed as the following mathematical expression:

$$\theta^{k+1} = \theta^k - \lambda \times \nabla C(\theta^k) \tag{8}$$

Where λ corresponds to a chosen learning rate and describes a step size of the algorithm. However, training suffers from overfitting, the phenomenon when the trained model explains well the examples from the training set, but it works relatively poorly on the examples that did not participate in the training (on the examples from the test set). In other words, overfitting in most cases occurs when the resulting polynomials have too large coefficients. Accordingly, this can be dealt with in a rather natural way by adding a penalty to the target function, which would punish the model for too large coefficients. Regularization is the most common method to solve overfitting problem.

Regularization is a method of adding some additional information during training phase to network parameters to prevent overfitting. This information often has the form of a penalty for the complexity of the model. For example, it can be constraints on the smoothness of the resulting function or constraints on the norm of vector space.

There are many ways to apply regularization in statistics and ML. We describe three most common and widely used, such as L_1, L_2 regularization penalty and dropout.

$\mathbf{L_1}$ regularization is a way to add L_1 distance term (absolute value of magnitude) to loss function as a penalty. It can be described with the following formula

$$L_1 = \sum_i C_i(\theta_i) + \lambda_r \sum_i |\theta_i| \tag{9}$$

where C is the cost function, λ_r is regularization coefficient and θ_i corresponds to the parameters of the network.

L_2 regularization applies 2-norm constraints to the cost function C as a penalty. It can be described with the following formula.

$$L_2 = \sum_i C_i(\theta_i) + \lambda_r \sum_i {\theta_i}^2 \tag{10}$$

Dropout is the method of regularization of ML models that is designed to reduce overfitting of the model. The essence of the method is that in the learning process, a layer is selected from which a certain number of neurons (for example 50%) are randomly emitted, further calculations are turned off. This technique improves learning efficiency and the quality of the result [16]. More trained neurons gain more weight in the network [40].

2.2.2 Activation functions

Activation functions are used to define the output of a neuron. We consider Rectified Linear Units (ReLU) and Softmax activation functions in this thesis.

ReLU is the activation function that takes only positive part of its argument. It can be described as the following formula:

$$f(x) = x^+ = \max(0, x) \tag{11}$$

where x is the input to the function. This activation function shows better performance in training of deep networks than the previous popular choices, e.g. hyperbolic tangent, logistic sigmoid [12].

Softmax is the activation function that converts a vector z of dimension K to a K-dimensional vector σ, where each coordinate σ_i of the resulting vector is represented by a real number in the interval $[0, 1]$ and the sum of coordinates is 1. Therefore, it is used to represent probability distribution and usually applied at the last layer of the network to get the probabilities as predictions. It can be described as the following formula

$$\sigma(z)_i = \frac{e^{z_i}}{\sum_{k=1}^{K} e^{z_k}} \tag{12}$$

2.2.3 Evaluating supervised ML models

The goal of any ML model is to learn general trends in data that will perform well on an unseen data. Once the training is done it is important to evaluate the performance of the model on new examples that were not used during the training of the model. There are several performance metrics. For supervised models, the most popular and simple metric is ***accuracy***. A model's accuracy is a fraction of right *predictions* on unseen test data (held-out data) that match the *actual labels* (ground truth data) to the total number of samples in the test set. It shows how well the model is able to generalize patterns in the data and not to memorize it. However, accuracy loses its practicality if the dataset is unbalanced. It no longer can show a true picture of how the model performs in a multi-class scenario where label distribution is uneven across several classes. In this case, the performance metric needs to treat each class equally and show how well the model performed on average no matter class distribution.

Precision corresponds to the ratio of true positive values (correctly labeled) and total number of true positives and false positives (incorrectly labeled) $\frac{tp}{tp+fp}$. It shows the classifier's accuracy when claiming a sample is a member of the positive class.

Recall corresponds to the ratio of true positive values and the sum of true positives and false negatives $\frac{tp}{tp+fn}$. It shows the classifier's capability to identify all positive samples.

F-score is the harmonic mean of precision and recall. F-score is calculated with the following mathematical expression:

$$F = 2 \times \left(\frac{precision \times recall}{precision + recall} \right) \tag{13}$$

F-score reaches its best value at 1 and worst at 0. Initially, F-score is for binary classification. However, it can be applied to multi-class scenario as well by calculating F-score values for each class with respect to the rest and average it across all classes (macro-averaged) F-score.

In this thesis we mainly use the macro-averaged F-score agreement (F-agreement) to weight each class equally and get a fair comparison that is independent of label distribution. F-agreement is calculated w.r.t the target model predictions as ground truth data and the attacker model as held out data.

2.3 Neural Networks

In this work, we define a neural network as a mathematical model, as well as its software or hardware implementation. It was originally designed on the principle of organization and functioning of biological neural networks - networks of nerve cells of a living organism. However, a neural network is also viewed as a parameterized function $f : X \to Y$ that produces some output $y \in R^m$ from some input $x \in R^n$. As the name suggests, neural networks contain several *neurons*. Each neuron in the first layer receives inputs from a set number of sources $x = [x_1, \ldots, x_n]$. These inputs are scaled with a parameterized weight vector and a bias term, the results of which is finally passed through an activation function $\sigma(\cdot)$. Therefore, the output of a neuron is $f(x, w) = \sigma(w \times x + b)$. Usually, neurons are combined into layers and layers into networks. Consequently, a deep neural network (DNN), is a hierarchical composition of large number of neurons and layers.

Convolutional Neural Networks (CNN) are DNNs that consists of convolutional neurons and layers. A convolutional neuron is a type of neuron that perform the convolution operation (Section 2.1.1) on the input [23]. The main difference from ordinary DNN is that neurons are arranged in several dimensions which makes it efficient to use CNNs on images. That is weights are represented as multi-dimensional matrices and denote as kernel of a convolutional neuron. The most common kernel is a two-dimensional matrix that can be written as 2D kernel. CNN that is using 2D kernels is referred as 2D convolutional CNN. There are also 1D and 3D convolutional CNNs, but these are not used in this thesis. The other difference is that in CNNs parameters are shared and the same depth slice of neurons is computed as convolution of the neuron's weights across the given input.

Typically a convolutional layer is followed by pooling layer to perform down-sampling of the input. The pooling layer is a unifying convolution neuron with a positive step h that aggregates input data in a rectangle area of size $h \times h$. *Maximum* and *mean* are typical aggregate functions. Convolutional layers may also consist of other parameters, such as padding and stride. Stride is the step size of convolutional kernel across the input. Padding controls the output dimensions of the layer. In this work we use two common padding types, "valid" and "same" in particular. "Valid" padding basically means that there is no padding at all. For example, if image dimensions are $n \times n$ and kernel size is $f \times f$, the new dimension would be $n' \times n'$, where $n' = n - f + 1$. In this case no additional padding is applied. However, "same"

padding convolution keeps dimensions of the output at the same value as it was in the input. In this case, padding value is always equal to $p = \frac{f-1}{2}$ where f is the convolution kernel size. Typically zeros are padded in the padding operation.

DNNs are typically used with back-propagation [23], a process that is used in gradient descent algorithm (Section 2.2.1) where errors are propagated through the network and the contribution of each parameter of the network is calculated with the chain rule (Section 2.1.3). This process allows training of a DNN to incrementally become better at its task.

2.4 Machine-Learning-as-a-Service platforms

Everything-as-a-service (XaaS) is a collective term to describe different as-a-service platforms such as Software-as-a-Service, Platform-as-a-Service, and Infrastructure-as-a-Service. XaaS concept allows mobilizing software across different networks, including ML software. It allows providers to offer a great variety of resources as cloud-based on-demand services to clients. In addition, it opens numerous possibilities to small businesses or individual entrepreneurs, who have a limited budget or resources for a variety of everyday tasks, to move their business to the cloud with high computational power.

For most parts, any technology that can be provided over the Internet and delivered on-site can be included in XaaS [38]. ML is not an exception and has a highly valuable part of this concept due to a nowadays increasing client-side demand for various tasks that need ML.

Machine Learning-as-a-Service (MLaaS) platforms are typically cloud-based infrastructures that provide web API to train, validate and use ML model. Usually, service providers try to construct their interface in a way that even a non-expert user can understand it. There are many different services for various clients [43].

2.5 Adversarial Machine Learning

Adversarial ML is a research area where ML algorithms and tasks are considered in the presence of an adversary. A clever adversary can modify the input to the model in a way that it compromises some aspect of overall system security, such as confidentiality, integrity or availability. For example, spam filters can be misled by the adversary and label spam message as non-spam or vice versa which will cause a violation of the system's ability to provide a service as intended.

In the domain of image classification or recognition, an adversary can apply a small change to the image in order to "fool" a target model in the output prediction. There are several techniques to craft such samples. In this work we use the two proposed techniques: Fast Gradient Sign Method and Projected Gradient Descent [13, 30].

2.5.1 Adversarial examples

An adversarial example can be defined as a modified sample of original data that is misclassified by a target ML model while retaining its functionality. Typically, samples are changed with a value (perturbation), so that the samples still very similar to the original image (measured with L_p distance Section 2.1.6). The definition of functionality is vague and depends on the use case of a ML task. To give a better understanding consider the following cases:

1. A malware detection software aims to recognize malware and viruses on given input files. An adversary's goal is to modify a malware sample in a way that it is misclassified by the detection system but the overall functionality of the malware remains the same. In other words, the adversary tries to solve a problem of finding a minimal perturbation applied on the sample that would maintain the intentional ultimate behavior, but also "fool" the detection system at the same time.

2. In image classification models, technically the perturbation size could vary a lot depending on how the system is used. Suppose, we have a system that performs image detection/classification and gives the input image to the user to verify prediction (i.e. online image recognition service, such as Google image search). In that case, the adversarial example would be verified by the human eye which puts some constraints on the adversarial examples crafting (typically in the form of a lower L_p norm). With that scenario, an adversary would need to find the smallest perturbation possible for it to be unnoticeable for the human eye and at the same time fool the detection system.

3. In many image detection/classification models, classification is done without human interaction or verification, such as in real-time prediction of objects from cameras, self-driving cars etc. In these cases, the perturbation applied on the image needs to mislead a target classifier.

Overall, the examples above represent so-called *white-box* scenarios in which an adversary has access to original model's internal outputs and can perform back-propagation to calculate gradients. Fooling a target model without access to model internals is considerably more difficult. However, this is the situation in many real scenarios: the adversary may only have access to a prediction API that returns probabilities or labels. That is the so-called *black-box* attack scenario. In this case, an adversary often needs to first "steal" a model and craft adversarial examples on the attacker model, to find examples that would be misclassified by the attacker model in the hope that the same adversary example would mislead the original model as well. These are referred to as *transferable* adversarial examples. Papernot et al. [35] studies the transferability property for adversarial examples. In general, the authors introduce an algorithm that exploits adversarial examples transferability in a black-box environment. They explore various algorithms, such as neural networks, logistic regression, support vector machines, decision trees, nearest neighbors, and ensembles and test transferability on the MNIST image dataset using model stealing techniques.

Fast Gradient Sign Method (FGSM) One of the techniques to craft adversarial examples was introduced by Goodfellow et al. [13] is the $L_\infty - bounded$ attack, which computes a perturbation at a given point in the direction of the gradient at that point. It modifies the original sample in the following way:

$$x + \epsilon \cdot sign(\nabla_x J(\theta, x, y)) \tag{14}$$

where θ are parameters of a model, x is input to the model, y is original labels that correspond to x, $J(\theta, x, y)$ is a cost function used to train machine learning model, ∇ operator is the gradient of the cost function J represented by the Jacobian, $sign$ is the operation of taking the sign of the input and ϵ is the size of perturbation. Therefore, this method implies that we calculate a gradient of the cost function with respect to the *input image*. This is similar to model training where we calculate gradient with respect to model parameters and modify them to minimize the loss while keeping input as constant. However, in FGSM we keep parameters constant and try to modify the input image to maximize loss. Afterward, FGSM takes the sign of Jacobian matrix of the result, meaning that Jacobian matrix values will contain -1 for all negative values and +1 for all positive values. Consequently, the distance to the original image is exactly ϵ. As can be seen from Equation 14, it is mandatory

to select some ϵ value that is the parameter that controls the size of perturbation. The resulting weight vector is added to the input image. The produced image x' is called an adversarial example, if the classification result $f(x)$ is different from $f(x')$. FGSM is an *untargeted* adversarial example crafting method. It does not result in the creation of a specific class, but only a class other than y. An example of such an image can be found in Figure 2.

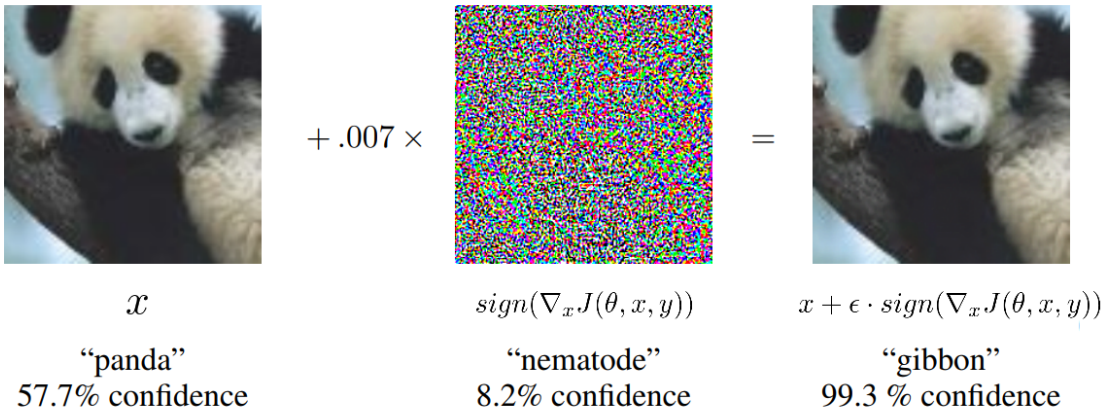

Figure 2: An illustration of adversarial image using FGSM [13].

Projected Gradient Descent Another technique introduced by [30] is PGD. PGD computes a perturbation using gradient decent in its constraint form iteratively changing the original image towards some target label. Assume a model outputs are some probability distribution $P(y|x)$ over labeled image data x. To craft adversarial example using PGD we try to find a new x' image where the loss function is maximized for a given target label y'. In other words, the output of the algorithm will be a new sample that would be misclassified as a target class. PGD constraints the norm of perturbation and can be used with various norms. Often it uses L_∞ box ($\|x - x'\|_\infty \leq \epsilon$) to control perturbation size such that the adversarial example is not much different from original sample.

Overall, PGD is a multi-step algorithm and repeats similar steps as FGSM until it converges or the number of steps is reached. One step in the L_∞-bounded PGD is shown below:

$$\begin{aligned} x' &= x' + \alpha \cdot \nabla_x J(x', y') \\ x' &= \ clip\ (x', x - \epsilon, x + \epsilon) \end{aligned} \tag{15}$$

where α is a step size in the gradient descent algorithm, *clip* is a clipping operation that puts all pixel values bigger than $x+\epsilon$ to be exactly $x+\epsilon$ and all pixel values less than $x-\epsilon$ to be exactly $x-\epsilon$. Therefore, from Equation 15 it can be seen that algorithm iteratively changing image x' towards class y'. PGD is a targeted attack. It results in the creation of an adversarial example that is misclassified as a specific class. An example of such image can be found in Figure 3.

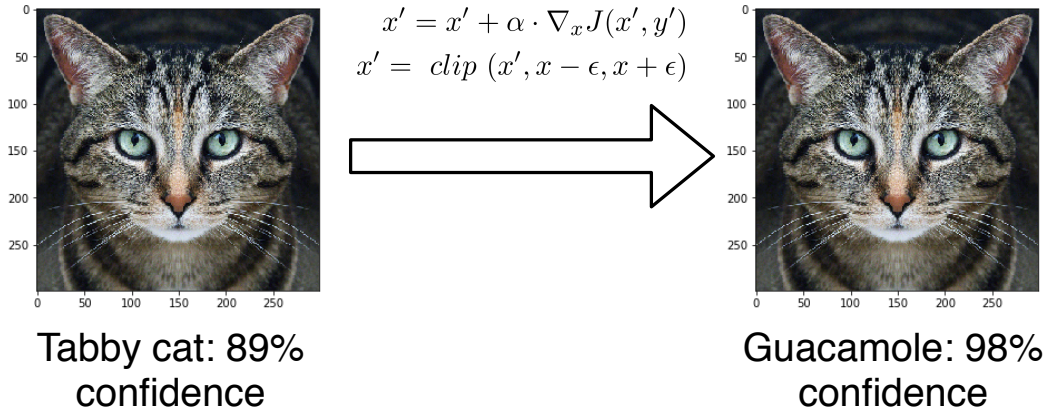

Figure 3: An illustration of an adversarial image using PGD method [30].

2.6 Model extraction attacks

Model extraction attack is a way of querying a machine learning model to exploit output information with the goal of violating model confidentiality. In this work, we consider two techniques for model extraction that have been introduced before. These are used as a baseline to compare the performance of our novel extraction techniques proposed in this thesis (Section 6). All techniques rely on training a local classifier and refining the classifier with output information from a target model.

2.6.1 Tramer attack

Tramer et al. [41] show several attack techniques to extract a model using prediction APIs from various cloud providers. Their technique aims for simple architecture models, such as logistic regression, decision trees, Support Vector Machines (SVM) and "shallow" neural networks. They demonstrate stronger attacks when model output confidences are available. For example, for logistic regression they propose to use an equation-solving attack due to the simplicity of the architecture. Another

technique is to extract decision trees by abusing confidence values as identifiers for paths. Both of these attacks show high extraction accuracy and required number of queries to steal a model depends on the number of input features, however they are not scalable and can only be applied to the simple models. They also propose three different ways of exploring decision boundaries of the original model. The first is to query random inputs and continually retrain a local classifier to match the label output from the original model. Secondly, they use line search techniques to find points close to decision boundaries of the original model and train a local model on those for several rounds. The third one is seen as the combination of two previous ones and was found to be the most powerful among these. It is called *Adaptive retraining* and starts with sending a first initial batch of uniformly distributed queries to the server model and training the local classifier based on those. Afterward, as in the second technique they try to explore the decision boundary of the local model and find points along the boundary, labeling data points with the original model. The process repeats for several rounds and the local classifier is retrained with new data each time. Tramer et al. [41] show that on tested datasets, such as handwritten digits (8x8 images), Adult dataset, Iris, Email Importance dataset etc. [5, 44], the attacks can achieve great result close to 100 % accuracy on the attacker model, but the attacks required a large number of queries. Even though those attacks produce considerably good results, they are only tested on shallow models and with low dimensional data. As a rule of thumb, at least $100 \cdot N$ queries (where N is the number of parameters) are needed to reach accuracy of 99 % on neural networks. N can go up to 500,000 parameters in common tutorial datasets used in ML research, for example MNIST, which results in 50 million queries to perfectly extract the target model. This is very high number of queries and therefore attack is not suitable for high-dimensional data with non-trivial structure.

2.6.2 Papernot attack

Papernot et al. [36] propose an attack strategy and a black-box way to craft transferable adversarial examples. The main idea is to locally build a new classifier that will reproduce the outputs of a server model to a certain extent. It is necessary to have some natural samples from original dataset or drawn from the same distribution as the original training set, at least one sample from each class. In the paper they use between 10 to 24 samples per class, depending on the dataset. From those initial

samples new synthetic data is generated based on the FGSM algorithm (Section 2.5.1) in order to get similar samples to original and to explore the decision boundaries of a server model by sending those to the target model. They call the method of generation a sytnthetic data Jacobian-based data augmentation. Later on, newly generated samples are used to train a substitute local model in several rounds. The synthetic samples are repeatedly verified with the original model. This way the local model learns the decisions of the server model more and more, such that it can replicate the model behaviour in a good enough way. Afterward, Papernot et al. apply white-box attack strategies to the local model to create adversarial examples and show that a portion of these are transferable. An advantage of this attack is that it has considerably low requirements in training data, almost no prior knowledge about the original model (only the nature of the task that the original model was trained for to select a suitable architecture for it). A major factor for an attacker is that the number of queries to the server should be low with the appropriate search strategies proposed in the paper. However, the techniques are done with the main goal of creating adversarial examples and not to reproduce the original model. They report the following results in local substitute training: on digit classification task (MNIST [25] test set) is 81.20% using 150 initial samples (15 samples from each class) and using a total of queries 6,400 to train the attacker model; they report the accuracy 71.42 % on traffic sign recognition task (GTSRB dataset [17]) using a 1000 initial samples (about 24 of each class), however the evaluation is somewhat biased as they use data from the same objects (see discussion in Section 6)

2.7 Technical background

In this subsection we present technical background, such as a brief description of the main library that is used throughout the work. We also give more detailed description of Intel Movidius NCS .

2.7.1 Pytorch

We use Pytorch Python package[2] to implement all solutions in this work. Pytorch is a package that operates with tensor object with strong GPU support. We choose Pytorch due to its popularity and computational speed compared to other packages.

[2]https://pytorch.org/

The main advantage is the use of dynamic on-the-fly compiler when building the DNN (unlike other frameworks, such as TensorFlow, Theano, Caffe).

2.7.2 Intel Movidius NCS

The Intel Movidius Neural Compute Stick (NCS) [18] is a USB fanless device for deploying deep learning applications at the edge without the usage of cloud-based services. The NCS is powered by low power high performance Intel Movidius VPU that is widely used in smart security cameras, drones or industrial machine vision equipment. The VPU uses 12 processors to accelerate computing by running parts of the networks in parallel. The NCS is used with Intel Movidius Neural Compute SDK that allows to profile, tune, and deploy DNNs on low-power devices that require real-time inferencing[3]. DNNs are trained using host machine and transfered to the NCS in a special graph representation via Neural Compute API (NCAPI). A graph network is attached to the VPU by NCAPI and is ready to be queried by the user. The output of the network is sent to the user via the USB connection and received by the application with NCAPI. What the output represents (labels or probabilities) is configured during a network graph generation and can only be changed by generating a new network graph.

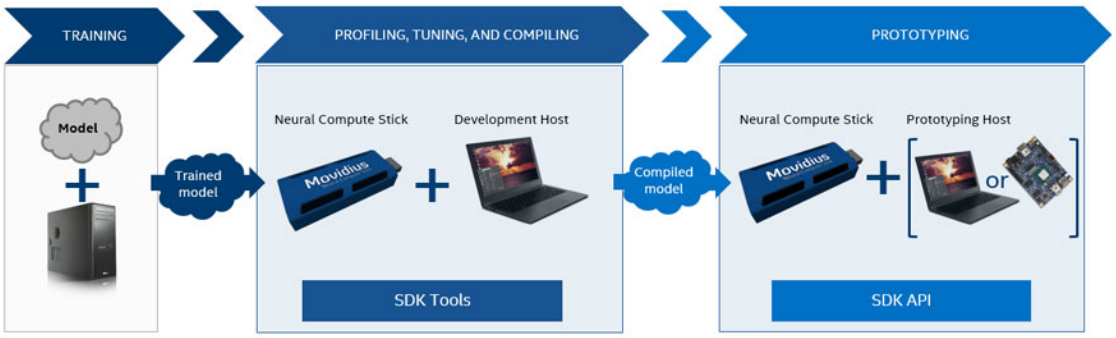

Figure 4: Intel Movidius NCS workflow

The NCS typical workflow example[4] is shown in Figure 4.

[3]https://developer.movidius.com/
[4]https://movidius.github.io/ncsdk/index.html

3 Problem Statement

3.1 Threat Model

The threat model of extraction attacks is described in this section. It consists of three parts: the attack surface, the attacker's capabilities and the attacker's goals.

Attack surface. An attacker can target any prediction API that responds with labels or probabilities to a given input. In this thesis we consider two following API scenarios:

Cloud-based prediction API. A malicious user interacts with a MLaaS platform, models protected by local isolation using trusted execution environment [34, 11] or by encrypted prediction schemes [27]. Systems themself are placed in a secure environment (e.g. cloud, network) and direct access to internal parts of the system are not allowed. The basic requirement of the attack is to have access to outputs of a target machine learning models. The attack is limited to neural networks. A simplistic scheme can be found in Figure 5.

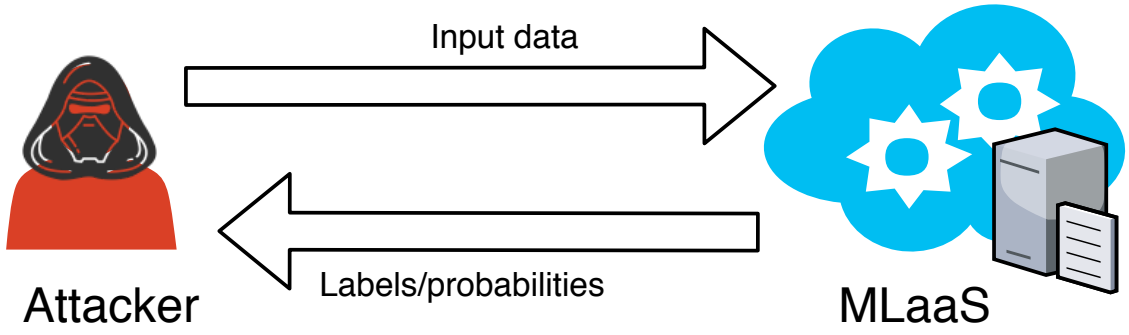

Figure 5: Scheme of cloud-based attack scenario

Hardware-supported prediction API. In this scenario, a model is embedded in some device. The server could have hardware-based protection, such as a TEE, e.g. Intel-SGX [39]. The model is intended to be used in real time, therefore, an attacker has several options to get predictions.

1. An adversary obtains a device if it is publicly available, for instance, an autonomous driving vehicle where the model should be provided, and tries to extract the model locally. Therefore, this scenario converges to the original attack described in the previous section. An adversary then could find suitable

adversarial examples and would theoretically have no limit on time. We evaluate this scenario in Section 6.3.

2. When no access to an API can be obtained, an attacker can still have physical access to the input sensor and can feed arbitrary data. Supposedly, an adversary would be able to send input as either physical printed images by showing it in front of the camera or by putting a screen in front where images would appear with a certain time gap. Undoubtedly, an attacker also would need to get access to decisions of the model somehow. This scenario is less plausible.

In Figure 6 we present a simple scheme of attack scenario on dedicated hardware prediction API.

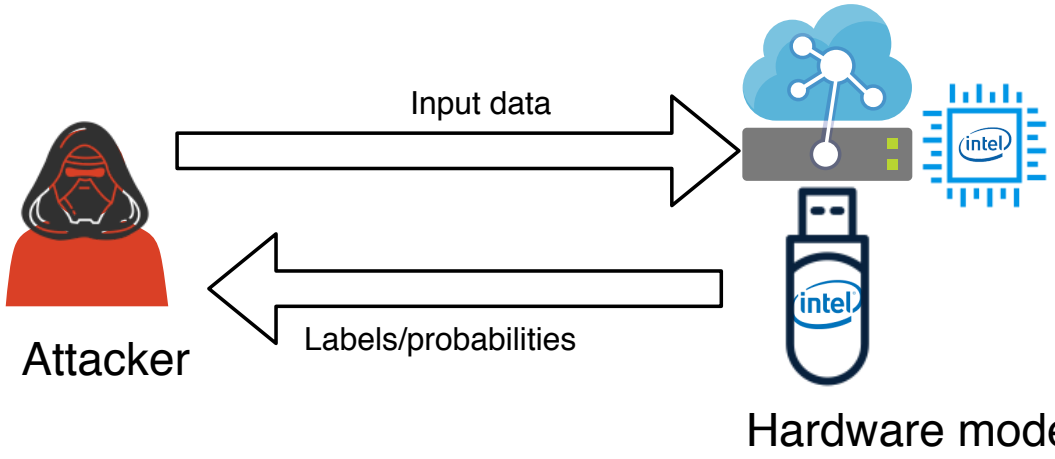

Figure 6: Scheme of hardware-based attack scenario

Attacker's capabilities. An adversary (Attacker) is assumed to have black-box access to the model (only access to prediction API). Attacker knows the nature of the input (classification task), shape of the input and output layers of the model. Attacker can send queries to the server and get probabilities or labels (or both) as a response. We assume that Attacker knows the model architecture, but not the parameters. He has no access to model internal data and processes as well as to the training data. The classes are presumed to be meaningful and correspond to a particular object of classification, i.e images, diseases etc. Attacker has access to a DNN training environment with commodity hardware. He is not limited in time, but in the number of queries he sends to the model.

Those assumptions are similar to the ones made in [36] and the main assumption is that Attacker should have access to some initial data from the same or similar

distribution targets as training data came from. However, assumptions made in this thesis are different from [27], which assumed that a malicious user has no intuition about its classification task, about what the input must look like, but rather only the shape of input and its domain.

Attacker's goals. The main objective of Attacker is to train a new model (attacker model) which closely resembles the target model based on classification performance on an unseen test set. Attacker has a budget limit and wishes to spend it as effectively as possible to "steal" the target model. The number of queries and their nature can be tracked by MLaaS service provider or any third-party service that is responsible for server maintenance. The secondary goal of Attacker is to create transferable adversarial examples, i.e to find perturbation ϵ for an input x of class c such that $x + \epsilon$ is classified as $c' \neq c$ by target model F. In other words Attacker selects a target class c' and uses algorithm from Section 2.5.1 to create new images $x + \epsilon$ from original images that are classified as c'

$$F(x + \epsilon) = F'(x + \epsilon) = c' \tag{16}$$

The perturbation size ϵ is bounded.

Therefore, Attacker tries to retain image appearance to be almost unnoticeable for the human eye. In contrast to previous work, we require that the attacker should be able to create targeted adversarial examples, i.e. misclassification into specific classes.

3.2 Requirements

The primary goal of this thesis is to critically evaluate existing model extraction techniques and to develop new techniques to extract a model from MLaaS platforms. The intended solution should be able to outperform existing attacks [36, 41]. Therefore, in the furtherance of the work done in this thesis, the solutions must fulfill the following requirements:

Implementation requirements

The implementation requirements serve as a basis for design decisions on the reproduction of existing attacks and developing new techniques using Python Pytorch library [37] that is rapidly developing and widely used in the ML community.

I1 Implementation of any method in this work must be done using Pytorch library from Python.

Performance requirements

The performance requirements are parts of improving on existing attacks. We state the following desiderata to evaluate this thesis in comparison with previous work. The quality of the solution is evaluated with regards to the following metrics.

P1 *F-agreement* In order for the attack to be considered successful an algorithm must produce a close approximation of target model. We aim to reach up to ∼20 % performance improvement compared to previous attacks.

P2 *Transferability rate* Another attack performance metric is how good the extracted model for crafting targeted transferable adversarial examples. We aim to reach up to ∼20 % performance improvement comparing to previous attacks.

P3 *Budget limit* The attacker has a budget limit and our technique must fulfill requirements *P1* and *P2* using fewer queries than state-of-the-art [41, 36].

P4 *Reliance of natural initial samples.* The proposed solution should rely on a few natural initial samples. Higher *F-agreement* and *Transferability rate* should be achieved using less initial samples compared to Papernot attack [36].

Scalability requirements

Our scalability requirements refer to the ability of the solution to be used in a range of applications. We evaluate attacks on dedicated hardware (Intel Movidius Neural Compute stick)

S1 Create an interface to communicate with Intel Movidius Neural Compute stick using the same implementation in Pytorch that corresponds to *Performance requirements* **P1 to P4**.

S2 Implemented API should be able to use the attack with several datasets.

4 Methodology

The previous section identified the problem scope of attacks against DNNs and discussed main threats that can arise due to a malicious behaviour. In addition, we identified the requirements for designing new techniques. Therefore, in this section we describe how we reproduced existing work on model extraction attacks. We also give an overview of new techniques to extract a ML model from a cloud service or locally deployed that fulfills these requirements.

Previous work in model extraction that is used in this work as a baseline is done by Papernot et al. and Tramer et al. [36, 41]. Tramer et al. evaluated the attack only using "toy" low-dimensional data and evaluating efficiency only based on the number of queries sent to the server. Papernot et al. expanded the attack to use more realistic images. However, the only parameter in the attack that Papernot et al. reported in evaluation is perturbation ϵ used in FGSM to generate synthetic data. In this work we identify and evaluate parameters in model extraction attacks that can improve extraction performance and catalyze the attack, such as different learning strategies (Section 4.1) and the amount of initial data the attacker has access to (Section 6.2.1).

We propose several novel techniques to query the target model. In general all approaches follow the same trend, which is crafting new synthetic data by querying the target model in some areas of interest. This may be going in the direction of the specifically chosen class or to opposite direction of the original class as in a previous solution [35].

The general scheme to extract DNN models can be systematized into the following Listing 1. We assume the target DNN classifier F and the attacker model F' that mimics F. Hyperparameters of F, such as the number of layers, the number of hidden neurons and activation functions are assumed to be known. These could be partially obtained using techniques from [33, 42]. We train F' using a minimum number of labeled training samples, as defined by a maximum prediction query budget b to F.

Model extraction algorithm can be described in the following way:

Listing 1: Steps in model extraction attacks.

1. **Choosing model hyperparameters.** Attacker selects a model architecture and hyperparameters to use for training his local model F'.

2. **Initial sample selection.** Attacker collects an initial set of unlabeled samples that build the foundation to catalyze the extraction attack, i.e. *the attacker set*. Samples are chosen depending on the the nature of the model input data (e.g. traffic signs or digit classification), and knowledge of what the output classes of the model mean (e.g. stop signs or number "9").

Duplication round

---

3. **Querying target model for prediction.** Unlabeled samples are sent to the target model in order to obtain labels/probabilities for them. These new labeled data are used as ordinary labeled data in next steps.

4. **Training local attacker model.** All labeled samples are used as a training set for the attacker model using a defined training strategy. We perform the intermediate evaluation of the attacker model after this step.

5. **Synthetic sample generation.** The method of generating new samples differs per technique. We use Jacobian-based sample generation technique to explore the input space in important areas. It is based on techniques for creating adversarial examples (Section 2.5.1). Newly generated data is then augmented to a dataset and used in step 3.

Steps 3 to 5 are looped until reaching some stopping criteria. It can either be until reaching adequate F-agreement value (**P1**) or until budget (**P2**) limit is reached.

4.1 Attacker model learning strategies

Previous work has not evaluated the impact of the learning strategy used by the attacker. The DNN background that is used in this thesis are described in Section 2.3, CNN in particular. CNNs requires several learning parameters optimized to achieve the best performance not only in training the model but also in the attack. Here we try to describe such parameters and give a general overview of their role in the attack.

Learning rate is the parameter used in gradient-based training to control the speed of learning on each iteration. It is well-known to be crucial in training of the model [23], and naturally will impact the model extraction attack (Section 4),

as it requires several rounds of training. The learning rate value also depends on regularization techniques used in training. Therefore, in this work we use a 5 fold cross validation scheme to estimate the right value for learning rate in different scenarios.

Regularization is usually used in training to avoid over-fitting and achieve better generalization of decision boundaries. It can be applied in training of the attacker model in several ways, such as L1 or L2-penalty and/or dropout (see Section 2.2.1). We found that the impact of L1 or L2 regularization penalty was small, but dropout tended to boost certain aspects of the attack.

Transfer learning relates to re-using machine-stored knowledge that was gained during previous training to solve the particular task and assign this knowledge to another related problem. One of the techniques that is used in transfer learning is to stop updating certain layers during back-propagation. This is called layer freezing [10, 26]. Therefore, during our first duplication round phase (Listing 1) model trained for one round might already have enough knowledge in intermediate convolutional layers such that it does not need further training. The benefit of layer freezing is faster training [10, 26]. CNNs convolutional layers tend to show poor ability to expand knowledge by taking new data bit by bit and, therefore we propose to "freeze" (stop updating them during backpropagation) all convolutional layers after first round of training on initial attacker set is done and for following duplication rounds we update only fully-connected (dense) last layers of the network, such as non-convolutional ReLU and Softmax layers (described in Section 2.2.2).

Model re-initialization In general, synthetic data generation technique can be described as incremental learning [19] as we feed more and more data to the model and re-train a classifier incrementally with the new data. CNNs tend to show poor performance in learning when new data introduced gradually over time and require the use of separate techniques to improve learning quality, such as feature extraction, fine-tuning, learning without forgetting [26]. Another problem in CNNs is the so-called "dying ReLU" problem [4]. ReLU outputs always zero when the input is negative therefore if during back-propagation part of the inputs become negative then those neurons are likely to be "dead" and have no effect in training. Sigmoid and Tanh activation functions can experience the same problems as their values

saturate. However, one solution to "dying ReLU" is to either use Leaky-ReLU or PReLU [15]. Both of these help only in long term and require a large set of data to "recover" a "dead" neuron in order to start learning again. Re-initializing all "dead" neurons with random values and allowing them to update normally according to gradient optimization algorithm used is highly time and power consuming. Thus, we propose a method to avoid the "dead" neurons and possibly improve learning experience by re-initializating model after each duplication round and train it from scratch, incrementally adding new synthetic data.

4.2 Synthetic sample generation

We use the attacker set, a subset of original data or data from the same distribution, to generate synthetic samples from labeled data. The goal is to increase the training set for the local classifier in order to better extract the target DNN model. Synthetic data generation is done in a way to explore target model's decision boundaries and get a better understanding of the classifier. We use the Jacobian matrix in accordance with the local classifier and update the model at each duplication round to improve the quality of crafted samples. To achieve better F-agreement with original classifier we explore the space in some directions of interest that we choose according to some specific query strategy. New synthetic samples are generated either with Fast Gradient Sign Method (FGSM) [13] or Projected Gradient Descent (PGD) [30], described in Section 2.5.1. We evaluated new approaches to find a suitable target label to craft synthetic data from the original set.

4.2.1 Jb-star approach

The idea in this approach is to query all possible target labels from a given class c, i.e $m-1$ directions, where m is the number of classes. By doing so we explore all possible decision boundaries of each class and expand the attacker model knowledge by querying all directions. The problem with this approach is that the synthetic data grows exponentially by factor m. Therefore, it takes a lot of time and resources to explore all directions. For instance, we retrain for N duplication rounds, therefore the total number of crafted samples would be $D_0 \times (m)^N$ where D_0 is the size of the attacker set that we start with. Even with simple multi-label classification task we need to store and send to the target model an enormous number of samples. Therefore, we decided to simplify this approach by exploring only some directions

further on.

4.2.2 Jb-Top-k approach

This approach is based on selecting target labels that are the closest to a given sample according to softmax probabilities from attacker's classifier. The reason behind selecting spatially closest classes is to attempt to get a better approximation of target classifier's decision boundaries. Consequently, selecting only the "best" target labels would save time and resources considering that exploring all directions would be too expensive.

The algorithm to find spatially closest targets to a sample x is following:

1. calculate probability distribution of the sample x which is $p_{1..m}$ w.r.t all classes $[1..m]$ using an attacker model

2. exclude the original class probability and sort the remaining probabilities from highest to lowest

3. afterwards, select the top-k classes and create synthetic samples with respect to these

Overall, as it was previously stated the amount of synthetic data will exponentially increase throughout duplication rounds by a factor of $k+1$. Therefore in our evaluation (Section 6) we choose k up to 3 in case of GTSRB and k up to 7 in MNIST, restricting the number of duplication rounds as well to keep the total number of queries within an acceptable range.

The top-k approach can be described using following equation:

$$x'_{c'} = x + \epsilon, \;\; s.t. \; \epsilon = \arg\max_{\epsilon} F'(x+\epsilon)[c'], \;\; c' \neq F'(x) \tag{17}$$

where c and c' are the original and target label correspondingly; F' is the local attacker model and $F'(x)$ is the class prediction from it; ϵ is a perturbation to the target class direction derived from attacker model using the Jacobian matrix and one of the techniques described in Section 2.5.1.

Consequently, in each duplication round the attacker set is augmented with new samples that the attacker model F' is likely to misclassify thus exploring areas of the input space that the model is the least certain about.

4.2.3 Jb-self approach

The concept of this technique is to apply methods from Section 2.5.1 using the same original label as the target. In consequence, it tries to give a better understanding of where part of original data lies, i.e. the most representative point of the class. It is done in accordance with the fact that the best way to reproduce original model's decision boundary is to train on the same data that the model was trained on.

Therefore, Jb-self can be viewed as the following formula:

$$x'_c = x + \epsilon, \;\; s.t. \; \epsilon = \arg\max_{\epsilon} F'(x+\epsilon)[c], \;\; c = F'(x) \tag{18}$$

where c and c' are the original and target label correspondingly; F' is the local attacker model; ϵ is a perturbation to the original (self) class direction derived from the attacker model using Jacobian matrix and one of the techniques described in Section 2.5.1.

4.3 Evaluation of model extraction

We evaluate the success of the attack using F-agreement and transferability rate.

4.3.1 F-agreement

F-agreement (see Section 2.2.3) is calculated with respect to the target model predictions as the ground truth data and the attacker model predictions as the held out data. We compare a number of predictions that models agree on to the total number of labels in data. It shows how well the attacker model agrees with the target model even though datasets may be imbalanced. We choose F-score agreement rather than prediction accuracy mainly to get a fair comparison of models and deal with class imbalance if any.

4.3.2 Transferability rate

Transferability was introduced by [24] (Section 2.6.2). We use a ratio of transferable examples to the total number of created adversarial examples as a performance metric called transferability rate. According to previous work (Section 2.6) transferability rate grows if the F-agreement between an attacker model and a target model increases. Adversarial examples can be targeted and untargeted. Targeted adversarial example refers to a sample that is crafted to be some specific (target) class that is different

from the original. Untargeted corresponds to a sample that is meant to be any other class but the original. The choice of which transferability rate is more appropriate to a certain case strongly depends on attacker's motivation and goals. Untargeted transferability may be enough to lower the overall prediction quality since it is faster to generate and requires less exactness in the attacker model in comparison with the target model. In another case, the attacker's goal may be to change all "Stop signs" to "Speed limits" or "Go straight" sign in this case targeted transferability is the appropriate choice for the attacker. In this thesis, we evaluate targeted transferability rate. We use an attacker set of initial samples to measure transferability rate. In case of MNIST, we target each sample to all other classes. In GTSRB there are too many classes to evaluate (42 directions to craft) so we propose a certain scheme to evaluate targeted transferability rate (see Section 5.2) by grouping several similar classes to a new macro-class and use it as a target. All of these attacks are also possible in a white-box environment where an attacker has access to a target model internal outputs, such as loss and back-propagation. In white-box scenario targeted and untargeted transferability rates are expected to be much higher.

5 Datasets and experimental setups

In the previous section, we described the methodology of our approach in general. This section presents an overview of datasets that are used throughout experiments and evaluation. Moreover, we describe two experimental setups used in this work.

For evaluation of proposed techniques in Section 4 and state-of-the-art techniques [35, 41] two main datasets are used in this work: MNIST and GTSRB. A brief overview of each dataset is presented below.

5.1 Mixed National Institute of Standards and Technology (MNIST)

MNIST dataset consists of 28×28 handwritten digits. It contains 10 classes (from 0 to 9). It is a subset of a larger set from NIST. All images were rescaled to fit in 20×20 boxes preserving their properties, centered and normalized to be gray-scale. MNIST dataset is constructed from NIST's Special Database 3 and Special Database 1. Special databases in NIST contain images of handwritten digits from different people. For example, SD-3 was accumulated from Census Bureau employees, whereas SD-1 was collected in a group of students. The resulting MNIST training set is built from 30,000 images from SD-3 and 30,000 images from SD-1. The test set consists of 5,000 images from SD-3 and 5,000 images from SD-1. Overall, the dataset is composed of images that were written by $\sim$ 250 distinct writers. Test set and training set are composed from different writers.

- Training set contains 60,000 images
- Test set - 10,000

Example inputs from each of the 10 classes from MNIST that vary within the same class are shown in Figure 7.

The number of images in each class in the test set is shown in Figure 8. Since the dataset is balanced, it is easy to reserve an equally divided set of samples from original data and use it as attacker set. Therefore, we reserve 5,000 (500 from each class) images from the training set, which are not used later on in training the target model. This dataset (or a subset of it) is used as the attacker set in our experiments with MNIST.

Figure 7: Example images from MNIST.

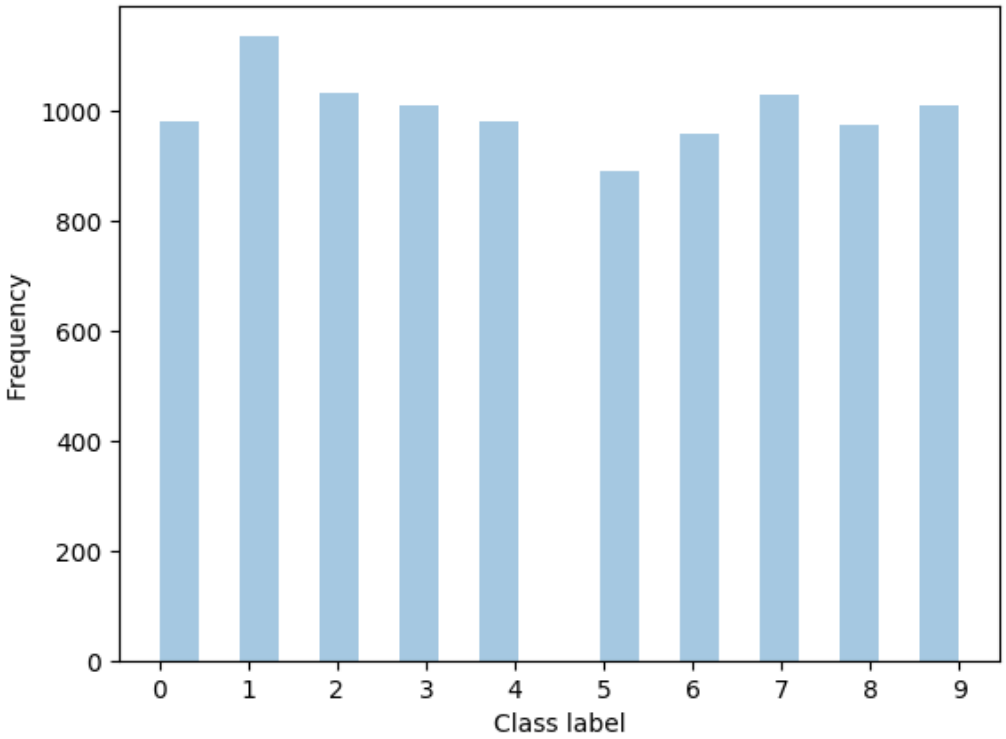

Figure 8: Distribution of classes in MNIST test set.

5.2 German Traffic Signs Recognition Benchmark (GTSRB)

The dataset GTSRB is downloaded from the official benchmark website [17]. It was collected as part of a competition for detection benchmark of traffic signs in IEEE International Joint Conference on Neural Networks. Initially, it consists of images with the size of 1360×800 pixels. Images contained all surrounded objects at that stage. After the competition, all images were cropped so that each image had exactly one traffic sign.

GTSRB dataset [17] contains 43 classes and more than 50,000 images in total. In the original state the images vary in sizes from 15×15 to 250×250 pixels, and traffic signs are not necessarily centered within the image. In this work, we crop images to be the sizes of 32×32 preserving their aspect ratio in order to feed the data to a DNN model, which only accepts fixed-size inputs. We further operate only with those sizes. GTSRB has following structure: training set contains 32,999 images, validation set - 6,210 and test set - 12,640 images. We present some examples of various classes in Figure 9.

Figure 9: Example images from GTSRB.

GTSRB dataset is very imbalanced and some classes are represented only with 60 images (Figure 11), e.g. class 0 - Speed limit (20km/h). Other classes contain up to 750 images per class (e.g. class 2 - Speed limit (50km/h)). This imbalance imposes certain restrictions on how to choose the attacker set for the attack. There is not enough data to both reserve enough samples in a separate attacker set from test set and exclude this reserved set from training/testing the target model. In addition to this, originally images of traffic signs in training set are of different sizes. Each class contains several images of the same traffic signs, e.g. "Speed limit (30km/h)", each taken from different angles and distances (30 shots of one sign). It can be seen in Figure 10. Consequently, it is not possible to reserve the attacker data from training set since then each image will cause excluding 29 other repetitions from the set as well. It results in an insufficient amount of data to train the target model and consequently in undertraining the target model. Therefore, in this work we "swap" the data when it was used for attacker data: the attacker set is sampled from test set distribution, and the evaluation of the attacker model is done on the training set. With that method of evaluation, an attacker does not use the data that server was trained on and evaluation is done on different data that attacker model is trained on. This is done only for the evaluating the similarity of the target and attacker model and is a fair evaluation strategy.

For easier evaluation of transferability rate, we identify macro-classes for GTSRB due to the following reasons. A large number of original classes (43) presented in

Figure 10: Structure of training set in GTSRB.

the dataset are very similar to each other, e.g. Speed limit (20km/h) and Speed limit (120km/h) and it is computationally heavy to evaluate all 42 possible directions for each sample in attacker set. We group traffic signs according to original shape and color. Signs are grouped as follows: (1) Warning signs, (2) Yield, (3) Stop, (4) Priority, (5) Red circle, (6) Blue circle, (7) Gray circle and (8) No entry. For example, this corresponds to the adversarial example using "Speed limit (120km/h)" as the target and crafted using a stop-sign as an initial example, is treated as successful transferable example if the target model classifies resulted adversarial image as one of the classes from macro-class (5) Red circle. We considered a transfer to be successful if target model predicts "Speed limit (30km/h)" or any other red-white circle that belongs to macro-class (5) Red circle.

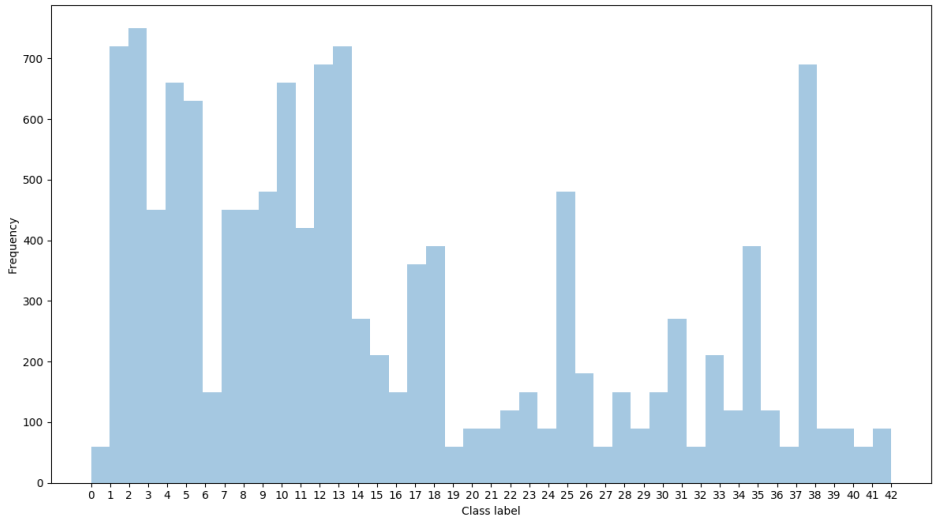

Figure 11: Distribution of classes in GTSRB test set.

Data preprocessing Initially, GTSRB is stored as images and MNIST is a serialized object that can be downloaded as an archive file from [25]. We save everything in Pytorch tensor format for convenience and normalize pixel values to be in the range of -1 to $+1$. This preprocessing is applied prior to sending data and target models are trained to recognize data in this range. The same range is applied for creating transferable adversarial examples as L_∞-bound as well as in the generation of synthetic samples.

5.3 Experimental setup

As mentioned in Section 3.1, model extraction attacks can target different platforms, e.g. cloud-based or local models. Next, we provide an overview of prediction API setups used in this work.

5.3.1 Ideal server

In the basic setup, Attacker is a remote user that sends input data to the server and gets labels or probabilities as a response. A simplistic scheme is presented in Section 3 in Figure 5.

In order to simplify the experimental setup, we simulate the MLaaS scenario in Section 3 with locally available (black-box) models, without involving any actual MLaaS platform. This setup avoids delays due to network traffic and can be seen as an ideal server that always responds to queries. Primarily, the MLaaS server is implemented as a file server that acts as a black-box model to the attacker's script and outputs either probabilities or labels, directly from the model. The server is implemented using Pytorch version 0.3.0. The server model is trained for 500 epochs using gradient-based training with a learning rate of 0.01. In GTSRB we additionally use dropout as regularization to achieve good classification results (i.e. good quality server model). The server model architectures for the datasets are described in Table 1. All max pooling is performed with "valid" padding type (described in Section 2.3). The architectures presented in Table 1 are used in this scenario only. The same architectures are used in the analysis by Papernot et al. [36]. We chose the same architectures in order to have an honest comparison of the state-of-the-art attacks and our new techniques.

Table 1: DNN model architecture for the ideal server scenario. The number of neurons is 471,120 in case of MNIST and 701,420 in case of GTSRB.

Layer	Type	Kernel	Maps & Neurons	
			MNIST	GTSRB
0	Input		1 map of 28×28	3 maps of 32×32
1	Convolutional	2×2	32 maps of 27×27	64 maps of 31×31
2	Max Pooling	2×2	32 maps of 13×13	64 maps of 15×15
3	Convolutional	2×2	64 maps of 12×12	64 maps of 14×14
4	Max Pooling	2×2	64 maps of 6×6	64 maps of 7×7
5	Fully connected	1×1	—	200 neurons
6	Fully connected	1×1	200 neurons	100 neurons
7	Output	1×1	10 neurons	43 neurons

5.3.2 Dedicated hardware-supported prediction API

In Section 3 in Figure 6 we present a simple scheme of attack scenario on dedicated hardware prediction API. For simulating a scenario with hardware embedded DNN model we use Intel Movidius NCS device with Intel VPU. Movidius provides an API that allows to query the device and get predictions from a certain model. However, queries can be processed by the API only one by one which slows down the attack (see Section 6.3). We deploy two models with following architecture where all convolutional layers are used with "same" padding type.

Table 2: DNN model architecture for a hardware-supported API. The number of neurons is 3,273,504 in case of MNIST and 1,832,400 in case of GTSRB.

Layer	Type	Kernel	Maps & Neurons	
			MNIST	GTSRB
0	Input		1 map of 28×28	3 maps of 32×32
1	Convolutional	5×5	32 maps of 28×28	108 maps of 32×32
2	Max Pooling	5×5	32 maps of 14×14	108 maps of 16×16
3	Convolutional	5×5	64 maps of 14×14	200 maps of 16×16
4	Max Pooling	5×5	64 maps of 7×7	200 maps of 8×8
5	Fully connected	1×1	1024 neurons	100 neurons
6	Output	1×1	10 neurons	43 neurons

Both models are trained for 200 epochs without dropout and using gradient-based learning with a learning rate of 0.0001. From Table 2 can be seen that model architectures for Movidius NCS are not the same as for ideal server. Models architectures are different than in the previous scenario due to challenges (**C2**) in transferring Pytorch models to Movidius NCS (see Section 6.4) and implementing exactly the same architectures. During this work, we tried several techniques to transfer Pytorch models on to Movidius device but most of them were unsuccessful and took a lot of time. The only viable solution we found is to train a server model from scratch (with TensorFlow) and deploy it using the API provided by Intel. Currently, Movidius NCS has several constraints on methods and functions from TensorFlow that are supported. For example, the type-casting operations are not supported or convolution operation may fail to find a solution for very large inputs. Therefore, we use MNIST model provided by Movidius NCappzoo package [5] that was trained according to TensorFlow tutorial on DNNs using MNIST dataset. For GTSRB we train the model using TensorFlow. Undoubtedly, for fair evaluation it is better to have the same architectures, but at the same time, we can record model extraction attacks behavior on wider models with more parameters.

[5] https://github.com/movidius/ncappzoo

6 Evaluation

We begin by discussing evaluation setups that are used throughout the evaluation. Next, we evaluate the impact of the attacker set size on our performance metrics based on the identified requirements in Section 3.2. Moreover, we look at the impact of the number of training epochs that the attacker uses, to isolate the impact of specific details of the attack itself, as opposed to previous work [36]. We, therefore, evaluate the impact of synthetic sample generation and incremental learning strategies by using transfer learning techniques: "freezing" layers and "resetting" model parameters. We evaluate the DNN model extraction attack targeting Movidius Neural Compute Stick (NCS). Finally, we list challenges that we met throughout the experiments in the evaluation.

6.1 Evaluation setup

Model extraction attack can be done for two purposes: to steal a target model and abuse monetary services using a local "free-of-charge" model or to create transferable adversarial examples for further malicious use. Therefore, we evaluate both in this work. We use two main performance metrics to assess the effectiveness of the extraction attack, the F-agreement and the transferability rate (see Section 4.3). Transferability rate is measured differently in MNIST and GTSRB due to the class imbalance in GTSRB. In MNIST targeted transferability rate is assessed according to the following pipeline:

1. Attacker selects d samples from each class in the MNIST attacker set. We chose value up to $d = 10$ samples per class in our evaluation depending on availability of samples in the attacker set. The attacker set size varies between 1 sample per class and 50 samples per class in our experiments.

2. Craft adversarial examples from the attacker set using PGD algorithm (described in Section 2.5.1) [30]. We use PGD algorithm to create adversarial examples against the attacker model F' targeting all classes except original label (in this case 9 out of 10 classes in MNIST) for each sample from the attacker set producing targeted adversarial examples $x'_{c'}$, where c' is the target label. We use a maximum perturbation $\epsilon = 128/255$ for MNIST. PGD is bounded by L_∞ – norm.

3. Adversarial examples are sent to the server model F regardless of their success at fooling F'.

4. Evaluate transferability rate success of $x'_{c'}$ as an agreement in classification by server model F as target class c' with $F(x'_{c'}) = c'$.

5. Aggregate results for all adversarial example combinations ($d \times N \times N - 1$), where N is the number of classes, and produce final transferability rate .

As stated previously we grouped original GTSRB classes into 8 macro-classes (see Section 5.2). Therefore, GTSRB targeted transferability rate is evaluated as follows:

1. Attacker selects d samples from each macro-class in the GTSRB adversary set. We chose value $d = 5$ samples per macro-class in our evaluation if that amount is available to Attacker. The number is lower than in MNIST due to constraints in speed and memory. The attacker set size varies between 1 sample per class and 50 samples per class in our experiments.

2. Craft adversarial examples from the attacker set using PGD algorithm (described in 2.5.1) [30]. We use PGD algorithm to create adversarial examples against the attacker model F' targeting $m - 1$ macro-classes (in this case 7 out of 8 classes in GTSRB) for each sample from adversary set producing targeted adversarial examples $x'_{c'}$, where c' is the target macro-class. We use a maximum perturbation $\epsilon = 64/255$ for GTSRB. PGD is bounded by L_∞ – norm

3. The adversarial examples are sent to the server model F regardless of their success at fooling F'.

4. Evaluate transferability rate success if $x'_{c'}$ as an agreement in classification by server model F as target macro-class c'_m with $macro(F(x'_{c'})) = macro(c') = c'_m$.

5. Aggregate results for all adversarial example combinations ($d \times 8 \times 7$) and produce final transferability rate.

We do not evaluate the performance of the attack in terms of speed of the attack, memory requirements, computational resource requirements etc. We only evaluate the performance in terms of adversary reaching ultimate attacker goals, i.e. model extraction.

6.2 Ideal Server: DNN extraction performance

We conduct several tests to asses that extraction attacks meet requirements defined in Section 3.2. Most of the evaluations are carried out using the ideal server (local Pytorch model) for speed and efficiency.

6.2.1 Attacker set size

We start with evaluating the impact of the attacker set size using several training strategies, such as training using probabilities output from the model, training with dropout and plain training with labels. At this experiment, we do not generate any synthetic data. We train classifiers (architectures from Table 1) for 200 training epochs with different numbers of initial samples available to Attacker. Evaluation sets are composed of non-overlapping sets that are used neither in training of the target model nor in the attacker set for MNIST and partially overlapping (training the target model) evaluation set in GTSRB.

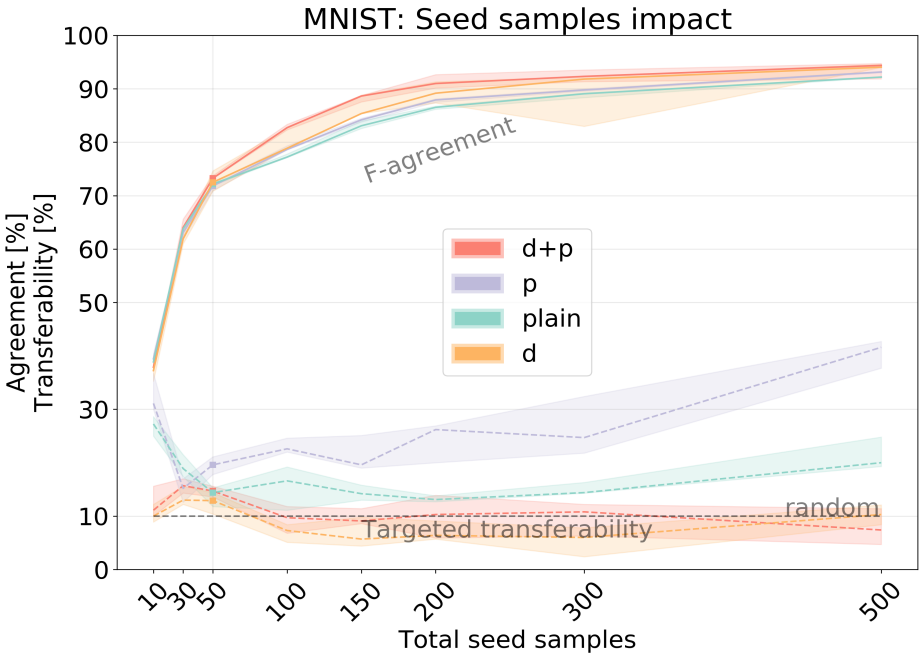

Figure 12: **MNIST target model**

We assume that seeing more original related data models improve the overall F-agreement since more training data that is highly related to the real distribution tend to improve the accuracy of the model, and we formulate a hypothesis that

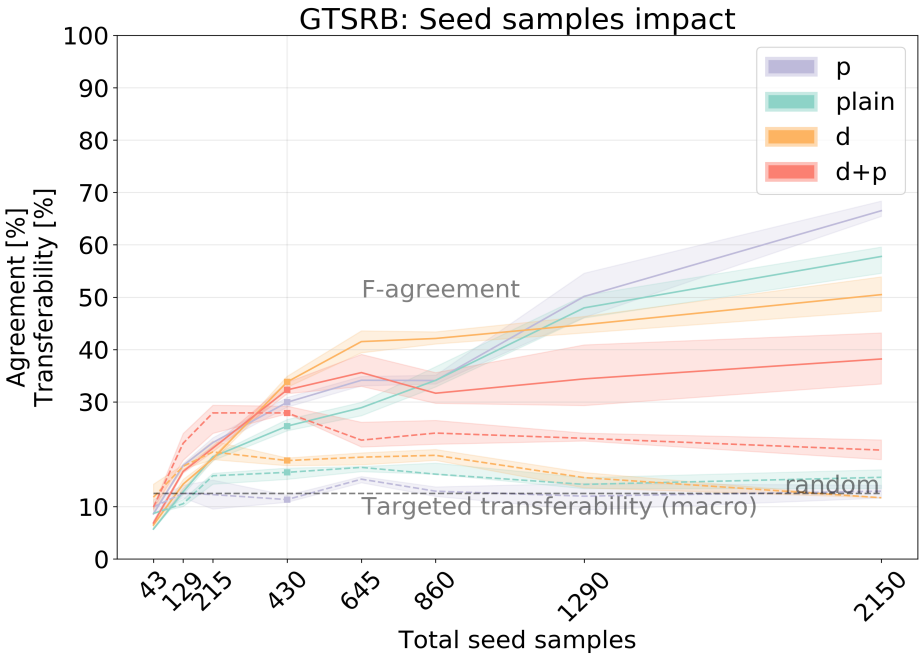

Figure 13: **GTSRB target model**

transferability rate improves with the number of real samples seen by the model so far. This can be rationalized with the fact that with more real samples decision boundaries of the attacker model would closely imitate target model behavior, meaning that it is easier to create targeted adversarial examples. The experiment is structured as follows: we train in total 32 different attacker models using four training techniques, such as the use of probabilities, dropout, dropout and probabilities, and plain training where neither of techniques is used. We select for each model a number of initial samples per class from 8 values (1,3,5,10,15,20,30 and 50). We present our experiment in Figures 12 (MNIST) and 13 (GTSRB) that feature median F-agreement(solid lines) and transferability rate (dashed lines), with using the following training strategies p=probabilities, d=dropout and plain=neither. It can be seen from Figures 12 and 13 below that overall F-agreement improves for all training strategies as was expected. It starts already with a high value of 38 % in case of MNIST using just one sample per class and going up to 93 % having 50 samples per class. Therefore, it may be sufficient to train an attacker model just on initial samples to get a good approximation of the target model without using synthetic data. In case of GTSRB F-agreement also does increase, having 8 % F-agreement with just one sample per class and going up to 65 % in the best case with having 50 samples per class. With

regards to the transferability rate, it can be seen that increase is small or none: the values dropped throughout the experiment, meaning our previous hypothesis was incorrect and *having more initial samples, thus better F-agreement, does not indicate improvement in transferability rate*. It might be related to the evaluation strategy of isolating attacker set from the original model and, thus eventually testing with data that the attacker model has not seen so far. Moreover, having an insufficient amount of initial samples per class, i.e. 1-3, shows fluttering behavior with a great variance. For instance, in MNIST with one sample per class, values vary between 10% to 30%. This can be justified with under-learning of the model and small sample size available for evaluating transferability rate having just 1-3 samples per class. Therefore, we select 5 initial samples per class for MNIST and 10 initial samples for GTSRB (50 in total for MNIST and 430 for GTSRB) to be the "optimal" starting point as a trade-off between having considerably low amount of initial data and acceptable starting F-agreement. In addition, it lies inside the so-called "elbow" region on the graph. Later on, we select the first five initial samples per each class from the attacker set and use it throughout the attack to evaluate performance. Furthermore, dropout and probabilities in GTSRB tend to show the best performance in transferability rate while worst in comparison with MNIST. We presume that this is due to the more complex structure of the GTSRB data and having 43 classes with a lot that look alike. This may introduce to PGD algorithm many alternative ways to reach a target class. One of the reasons that dropout performs better in GTSRB may be the fact that GTSRB target model is trained with dropout and MNIST target model without.

6.2.2 Training time

In the second experiment, we evaluate the impact of changing the training time for the attacker model to choose the appropriate value of training epochs for further experiments. We train an attacker model from scratch for one duplication round, therefore no synthetic data is generated, and each time we use a different number of training epochs. Figures 14 and 15 show median F-agreement (solid lines) and transferability rate (dashed lines) w.r.t. the number of training epochs for two target models (Section 5.3) using MNIST and GTSRB datasets (Section 5), where *plain* is a training strategy without using dropout or probabilities.

It can be seen in Figure 14 that F-agreement graph sharply increases from ∼65

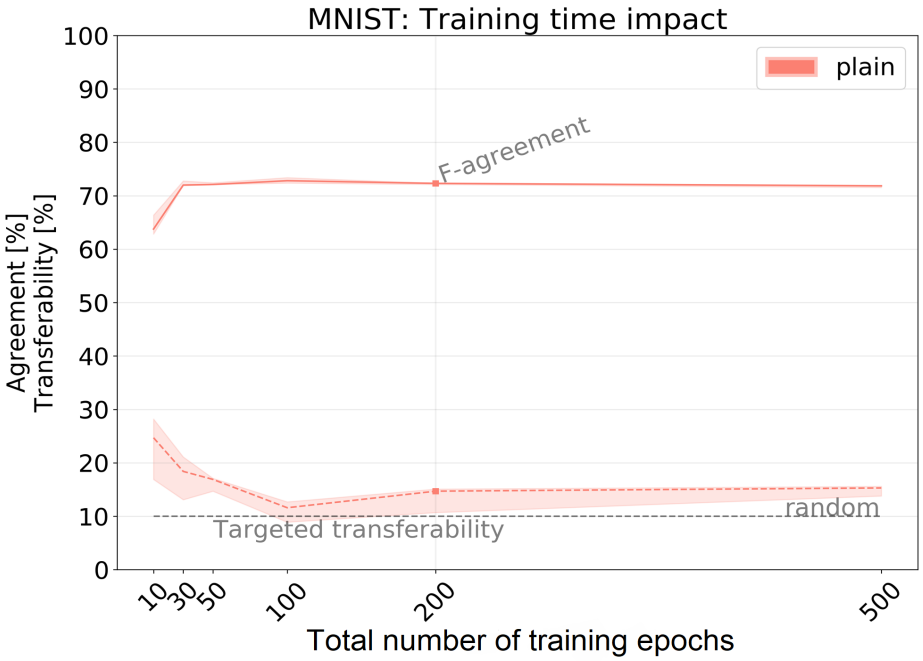

Figure 14: **MNIST target model**

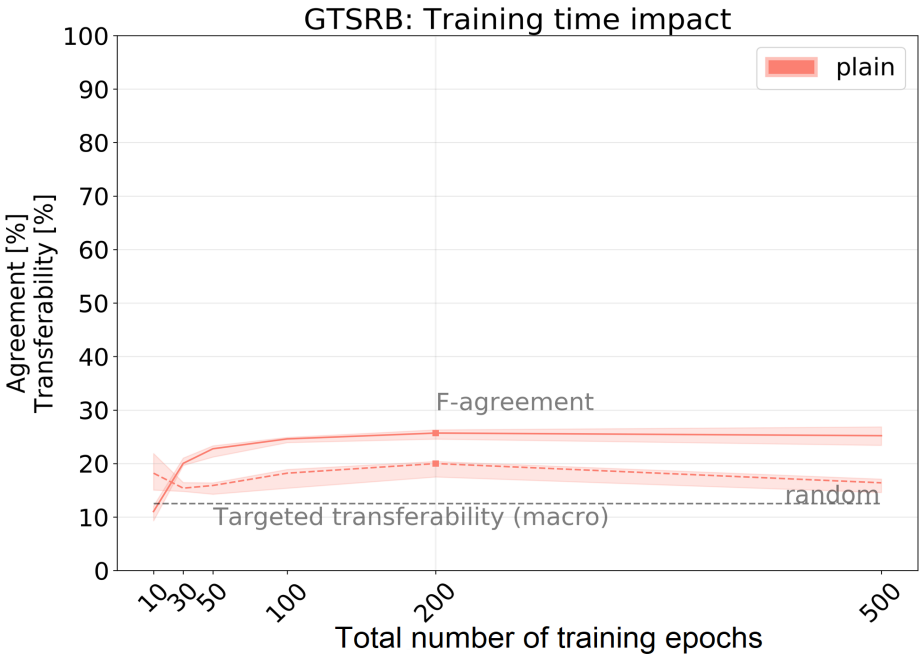

Figure 15: **GTSRB target model**

% to ∼73 % when using 10 and 30 epochs correspondingly in MNIST. However, afterward F-agreement remains stable and the graph does not show a clear impact

of having more epochs. Therefore, run-time can be saved by using fewer epochs in future if needed. Transferability rate shows unusual performance having bigger value at 10 epochs and slightly decreasing with more training applied. GTSRB in Figure 15 shows similar performance as in MNIST. Transferability rate also starts with one of the highest values and slightly goes down. However, unlike MNIST it recovers at 200 epochs and, therefore, we can identify the "clear" maximum point in F-agreement and transferability rate at 200 epochs. We will use the value of 200 epochs in our next experiments.

6.2.3 Comparison with state-of-the-art

In the third experiment, we look into the impact of synthetic sample generation on accuracy and transferability rate. We evaluated all presented training strategies with having probabilities and dropout, probabilities, dropout and just plain training and selected only four combinations of *Jb top-k* and *Jb-self* with different learning strategies according to the previous experiment. The reason that we limit the number of experiments presented is due to time and memory constraints. We choose k value according to the final number of queries Q to be equal at the last duplication round. In this case, we selected k to be $k \in [1, 3, 7]$ for MNIST and $k \in [1, 3]$ for GTSRB. We recall from Section 4.2.2 that the total number of queries is $Q = D_0 \times (k+1)^N$. Therefore, we choose k value always to be so that $k+1$ is 2^t, where t is some number. N number of duplication epochs also have an impact on the total number of queries and we adjust it accordingly. For example, assume $D_0 = 50$ and $k = 1 \implies Q = 50 \times 2^6 = 3200$. For $k = 3$ number of duplication rounds is 3 so that $Q = 3200$ and so on. In addition, we present results of state-of-the-art attacks, i.e. Papernot [36] attack evaluating with the same algorithm as in [36] adjusting some settings according to our experiment chosen values. They use 6 duplication rounds and train an attacker model for 10 training epochs in each round with a learning rate 10^{-2}. They use $\epsilon = 0.1$ to generate additional synthetic data. We evaluate Tramer attack using adaptive retraining strategy from [41], as a baseline to compare performances. We expect the F-agreement to have a major boost from synthetic queries, according to our description of the method Listing 1 we explore input space in directions of most interest to the model. We hypothesize that the transferability rate would improve over the substitute epochs as well due to the same reasons as for the F-agreement.

To generate synthetic data we use PGD technique (see Section 2.5.1).

6.2.4 Impact of ϵ value.

To motivate our choice of ϵ value in PGD for generating new synthetic data we look at the attack behaviour with different ϵ values. For MNIST dataset we use *Jb-top-7 p* attack with 3 duplication rounds and 3,200 total queries to the target model. In case of GTSRB, we use *Jb-top-3 d p* attack with 3 duplication rounds and 13,760 total queries to the server model. We present Tables 3 and 4 below showing performance in MNIST and GTSRB with having different ϵ value. We select epsilon values from a set $E \in [0.01, 0.1, 0.15]$. Highlighted values in bold are the best results obtained during this experiment.

Table 3: Attack results for different ϵ values in MNIST

ϵ	**F-agreement**	**Transferability rate**
$\epsilon = 0.01$	$71.7 \pm 0.7 \rightarrow 73.6 \pm 0.4$	$18.4 \pm 3.6 \rightarrow 22.1 \pm 7.5$
$\epsilon = 0.1$	$72.0 \pm 0.7 \rightarrow 73.3 \pm 0.4$	$19.8 \pm 1.1 \rightarrow 24.2 \pm 5.4$
$\epsilon = 0.15$	$\mathbf{72.8} \pm 0.6 \rightarrow \mathbf{88.8} \pm 1.2$	$\mathbf{18.2} \pm 3.8 \rightarrow \mathbf{54.6} \pm 5.5$

Table 4: Attack results for different ϵ values in GTSRB

ϵ	**F-agreement**	**Transferability rate**
$\epsilon = 0.01$	$\mathbf{35.5} \pm 4.8 \rightarrow \mathbf{55.1} \pm 2.1$	$29.3 \pm 3.1 \rightarrow 27.4 \pm 2.8$
$\epsilon = 0.1$	$34.9 \pm 4.0 \rightarrow 15.4 \pm 0.9$	$30.9 \pm 1.8 \rightarrow 54.6 \pm 6.9$
$\epsilon = 0.15$	$34.4 \pm 1.9 \rightarrow 10.5 \pm 1.1$	$\mathbf{28.5} \pm 2.9 \rightarrow \mathbf{59.9} \pm 2.9$

From Table 3 it can be clearly seen that smaller values in MNIST result in the lower final F-agreement and transferability rate results. Therefore, we select $\epsilon = 0.15$ for future experiments in MNIST as the optimal value. On the contrary, results for GTSRB (Table 4) are much different. Low $\epsilon = 0.01$ results in better F-agreement at the end of the attack but the transferability rate slightly drops. Whereas bigger ϵ values (0.1 and 0.15) show a dramatic fall in F-agreement and a huge rise in transferability rate. As a trade-off, we select $\epsilon = 0.01$ to have considerably good extraction F-agreement, since transferability rate remains almost the same. However, if the primary goal of the attacker is to create transferable adversarial examples then ϵ value needs to be bigger as we can see from the table.

6.2.5 Synthetic query impact

As it was stated before, we select perturbation size $\epsilon = 0.15$ for MNIST and $\epsilon = 0.01$ for GTSRB in PGD. Those values correspond to L_∞ bound with maximum perturbation of $0.15 \times (pixel_{max} - pixel_{min})$, i.e $0.15 \times (1 - (-1))$ or 19 pixels (7.5% of pixel range) for natural image range. For GTSRB ϵ is around 1.3 pixels (0.05% of pixel range).

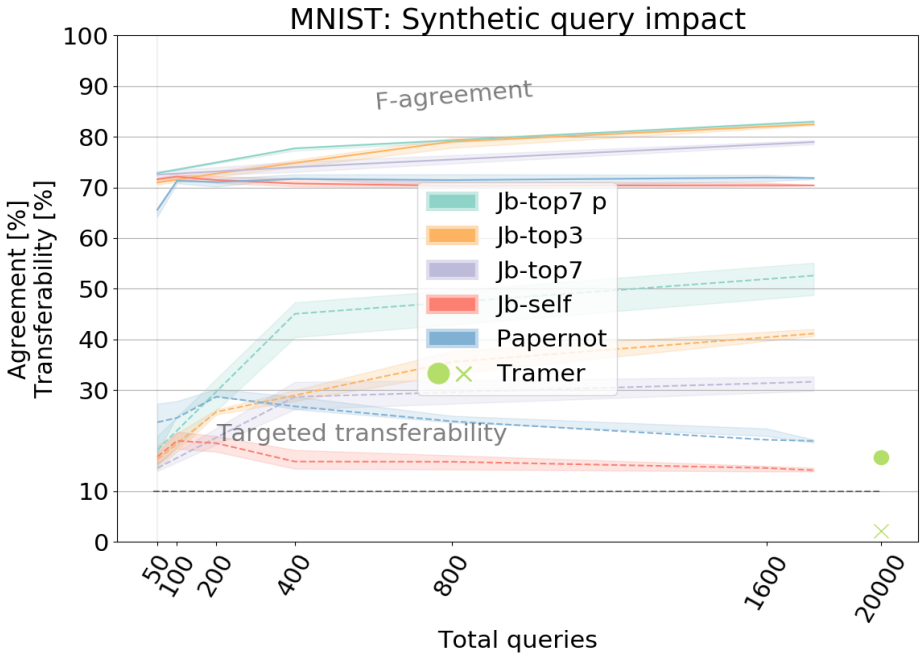

Figure 16: **MNIST target model**

Figures 16 and 17 demonstrate median F-agreement (solid lines) and transferability rate(dashed lines) w.r.t. number of queries (MNIST: 50 initial samples + rest synthetic, GTSRB: 430 initial samples + rest synthetic) evaluating the impact of synthetic samples generation techniques for two target models (see 5.3) using MNIST and GTSRB datasets (see 5), where *p=probabilities, d=dropout* and *Jb-topk/Jb-self/Papernot/Tramer* are query strategies. Randomly perturbed transferability rate is presented used dashed black line in the graph. It can be described as $1 - \frac{1}{m}$ transferability rate, where m is the number of classes. Random perturbations do not have a bound on perturbation size, meaning that all pixels can be arbitrarily changed.

We report Tramer attack performance as a green ball for F-agreement and as a green cross for transferability rate. It requires 20,000 queries to achieve low results

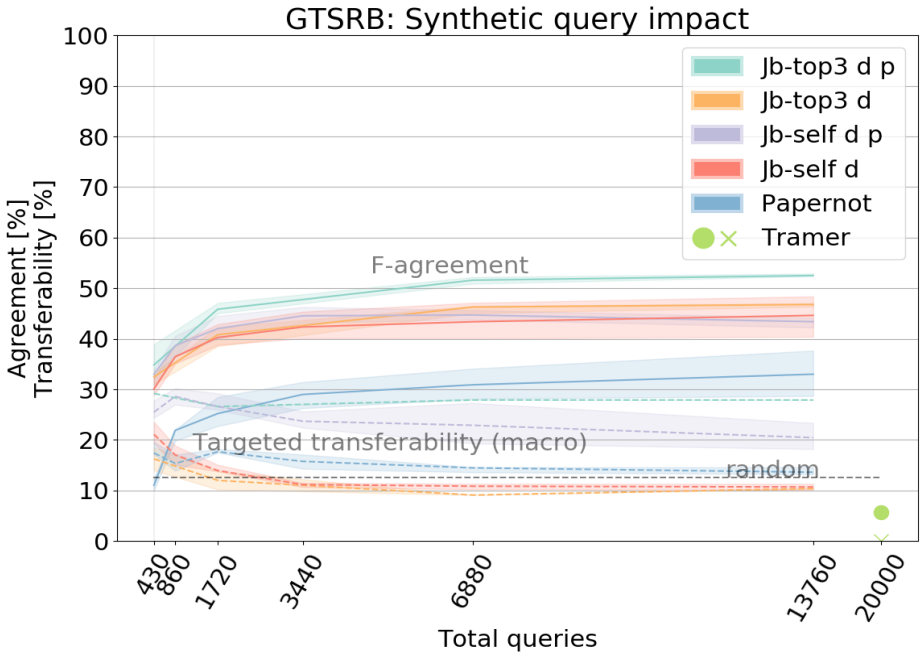

Figure 17: **GTSRB target model**

and therefore we plot the results as points on the graph so that it is readable. It can be clearly seen that Tramer attack shows poor performance for both datasets. Tramer attack also uses a significant amount of queries compared to other techniques to achieve slightly better than random results in agreement achieving on average 17 % F-agreement in MNIST and 7 % in GTSRB. However, transferability rate results of Tramer attack are worse than randomly perturbed images and achieve at the best 2.3 % transferability rate in MNIST and less than 1 % in GTSRB. This demonstrates that Tramer attack is not suitable for DNN models and only applicable to simple models described in [41]. In Papernot attack, we choose substitute training parameters according to the description in [36]. We restrict the number of queries in all of the attack (except Tramer) to 3,200 queries in MNIST and 13,760 queries in GTSRB. Papernot attack reaches on average 70 % F-agreement in 3,200 queries for MNIST dataset and 35 % in GTSRB using 13,760 queries.

Furthermore, we observe a sharp increase in performance using *Jb-top-k* and *Jb-self* approaches compared to previous attacks. F-agreement steadily increases for almost all techniques presented, with the exception of *Jb-self plain* in MNIST. *Jb-self* querying strategy is the most similar to Papernot attack and we can see that without any additional learning strategies applied it tends to show similar behavior. Overall,

Jb-top-k and *Jb-self* start with roughly 75 % using just 50 queries reaching 88 % on average in MNIST. The fact that we train for more epochs the attacker model gives a higher value in F-agreement already on the first epoch comparing to the previous attack. This shows that F-agreement is better for up to 18 pp. compared to the previous attack [36] by the last duplication round. In GTSRB we observe significantly smaller numbers due to more complex nature of the data in the dataset. *Jb-top3 d p* starts with F-agreement 30-35 %, and it goes up to 45-54 % in GTSRB. However, *Jb-top3* and *Jb-self* with different training methods are still better than previous attack [36] for up to 20 %. It can be clearly seen that impact of synthetic data generation is not as important as we expected. Figures show that an attacker can stop querying at already 400 queries in MNIST or 6880 queries in GTSRB since further querying does not give a large improvement in either metrics. Therefore, we conclude that *synthetic queries do not have a momentous impact on the F-agreement and demonstrate just a slightly upward trend.* Moreover, it can be seen that the use of probabilities is crucial in terms of good F-agreement and transferability rate performance.

With regards to transferability rate, we observe contradictory results. Transferability rate shows a significant improvement in performance with the use of probabilities comparing to dropout or plain, whereas dropout methods tend to have a negative impact on transferability rate and graphs show a slight fall. In MNIST best method with using probabilities achieves almost 55 % in targeted transferability rate which is a huge improvement of almost 35 % compared to previous technique [36]. It can be seen that even without access to probabilities we still see a moderate increase in transferability rate in *Jb top-k* methods and achieve up to 42 %. However, *Jb-self* demonstrates a slight drop.

In GTSRB we observe that all methods with dropout (no probabilities) show a dramatic drop from 15-20 % to 10 % and result in worse than random perturbation performance in transferability. However, as we mentioned before probabilities have a positive impact on transferability rate and overall results are better than with any other training strategy. We achieve up to 30 % transferability rate in the best case using *Jb top-3 d p*. The previous experiment (see Section 6.2.4) showed that transferability rate can be much higher with bigger perturbation size during the synthetic sample generation. However, with bigger ϵ suffers F-agreement. Therefore, if the goal of the attacker is to create transferable adversarial examples then F-agreement can be sacrificed in the favor of bigger transferability rate.

In general, *Jb top-k* shows good performance in both metrics that we used, whereas *Jb-self* either remains on the same level or demonstrate a vast slump. Therefore, synthetic queries seem to have minor impact on F-agreement and the improvement is small with the respect of the number of queries the attacker uses. In addition, we can see that the attacker does not need to have a good F-agreement in order to achieve good results in transferability.

Does a low number of training epochs help in transferability? As it can be seen from discussion and Figures 14 and 15 in Section 6.2.2 transferability rate starts at a higher value when the attacker model is trained for 10 epochs and decrease with having more training epochs. In this experiment, we selected two attacks *Jb-top7 p MNIST* and *Jb-top3 d p GTSRB* with the same experimental settings as in previous subsection, but we use 10 training epochs in each duplication round. We used 3 and 4 duplication rounds in MNIST and GTSRB correspondingly and repeat each attack five times. Below in Table 5 we show results for having 10 training epochs for the attacker model using *Jb-top7 p* MNIST and *Jb-top3 d p* GTSRB attacks, which were the best in the the previous experiment. Tables show mean performance at first duplication round with standard deviation and mean performance at last duplication round with standard deviation.

Table 5: Attack results using 10 epochs

Dataset	F-greement	Transferability rate
MNIST (10 epochs)	$63.4 \pm 4.4 \rightarrow 73.1 \pm 1.9$	$22.8 \pm 7.7 \rightarrow 33.4 \pm 3.1$
MNIST (200 epochs)	$\mathbf{72.8} \pm 0.6 \rightarrow \mathbf{88.8} \pm 1.2$	$\mathbf{18.2} \pm 3.8 \rightarrow \mathbf{54.6} \pm 5.5$
GTSRB (10 epochs)	$10.0 \pm 1.6 \rightarrow 25.8 \pm 1.8$	$16.6 \pm 1.9 \rightarrow 20.4 \pm 3.8$
GTSRB (200 epochs)	$\mathbf{35.5} \pm 4.8 \rightarrow \mathbf{55.1} \pm 2.1$	$\mathbf{29.3} \pm 3.1 \rightarrow \mathbf{27.4} \pm 2.8$

It can be seen from Table 5 that final results are worse than in the previous experiment even though starting values are bigger in some cases. Whereas F-agreement suffers and shows worse performance. Therefore, it is neither beneficial in terms of transferability rate or F-agreement for an attacker to under-train the model.

6.2.6 Impact of learning strategies

In this subsection, we present a comparison of different learning strategies for an attacker model that we hypothesize may help to improve performance of the extraction

attack. We select only one configuration from the previous experiments (Section 6.2.5) which are *Jb-top3 d p* in GTSRB and *Jb-top7 p* in MNIST. We present results in tables 6 and 7 below. All tables show mean performance at first duplication round with standard deviation and mean performance at last duplication round N with standard deviation. N is 3 in case of MNIST and 4 in GTSRB as in the previous experiment for *Jb-top7 p* MNIST and *Jb-top3 d p* GTSRB.

Table 6: MNIST performance comparison using different training strategies

Training strategy	**F-agreement**	**Transferability rate**
Standard training	**72.8** $\pm$ 0.6 $\rightarrow$ **88.8** $\pm$ 1.2	**18.2** $\pm$ 3.8 $\rightarrow$ **54.6** $\pm$ 5.5
Layer freezing	72.9 $\pm$ 0.6 $\rightarrow$ 74.7 $\pm$ 1.8	18.3 $\pm$ 2.4 $\rightarrow$ 31.7 $\pm$ 5.2
Resetting all weights	72.7 $\pm$ 0.5 $\rightarrow$ 72.5 $\pm$ 1.0	18.2 $\pm$ 3.9 $\rightarrow$ 25.5 $\pm$ 3.3

Table 7: GTSRB performance comparison using different training strategies

Training strategy	**F-agreement**	**Transferability rate**
Standard training	**35.5** $\pm$ 4.8 $\rightarrow$ **55.1** $\pm$ 2.1	**29.3** $\pm$ 3.1 $\rightarrow$ **27.4** $\pm$ 2.8
Layer freezing	34.5 $\pm$ 4.5 $\rightarrow$ 55.7 $\pm$ 1.5	29.9 $\pm$ 2.5 $\rightarrow$ 26.0 $\pm$ 2.6
Resetting all weights	34.0 $\pm$ 5.6 $\rightarrow$ 51.6 $\pm$ 0.3	30.4 $\pm$ 2.7 $\rightarrow$ 27.1 $\pm$ 2.3

Freezing convolutional layers during training As we discussed in Section 4.1 in here we freeze all convolutional layers after the initial training on attacker set, and after that, we train only dense layers of the networks. In MNIST (Table 6) we can see that freezing convolutional layers in the attacker model result in a huge decrease in F-agreement and in transferability rate in comparison to the **Standard training**. Therefore, we conclude that layer freezeing in the network have a negative impact on both performance metrics. For example, F-agreement stops at around 75 % with freezing whereas without freezing it goes up to 88 %. The same trend can be seen in transferability rate as it goes up to only approximately 32 %, whereas without layer freezing the attack achieves 54.6 % transferability rate.

In GTSRB (Table 7) we can see that layers freezing produce very similar results to the previous experiments. We see exactly same trends as it was in Section 6.2 that F-agreement goes up to around 56 %. Transferability rate also follows the same trend from the previous section, which is a steady decrease from 30 % to 26 %.

Resetting all weights in network at new duplication rounds According to the discussion in Section 4.1 we reset the attacker model before each duplication round and train it from scratch on growing attacker synthetic set. In MNIST (Table 6) we can see that resetting model parameters in the attacker model results in the F-agreement remains steady and stays at the same level of 73 % after 3200 queries. While without model resetting it reaches up to 88 %. Moreover, transferability rate show a huge difference in the final rate in comparison to standard training that is even worse than with layer freezeing. Therefore, model resetting obviously shows worse results than the previous techniques from Section 6.2.

In GTSRB (Table 7) we can see that resetting model parameters in the attacker model results in the F-agreement to reach only 51.6% value. When in fact, transferability rate is very similar to layers freezing and, therefore, as we noted before, it follows the same trend as in the previous experiments in Section 6.2. That indicates that initial idea of model resetting to mitigate "dead neurons" cannot be applied to all CNN models. It may be due to the width of the models (number of parameters in layers) that we used in the experiments.

Overall, we can note that both incremental learning (see Section 4.1) strategies resulted in worse or same performance in terms of F-agreement or transferability rate. Moreover, for MNIST our previous hypothesis made in Section 4.1 is partially wrong and model re-initialization as well as layers freezing do not help in learning and have almost no impact on GTSRB.

In conclusion, observed learning strategies show considerably poor performance in MNIST producing even worse results than before, whereas GTSRB dataset seemed to be almost unaffected by those learning strategies and results remain the same or slightly decrease. The reason behind the failure of incremental learning techniques might be due to an insufficient amount of data on each round after these techniques are applied. Incremental learning has proved to be efficient when a significant amount of data is fed to the model after re-initialization or layer freezeing [26].

6.3 Hardware prediction API: Movidius Neural Compute Stick

We evaluate the performance of the attack using hardware-based local model, i.e. Intel Movidius NCS [18]. We selected best techniques for two datasets according to results in Section 6.2. These are *Jb-top3 d p* for GTSRB using 13,760 queries in

total and *Jb-top7 p* for MNIST using 3,200 queries in total. During the evaluation of Movidius NCS, we met several challenges regarding run-time. For example, normally the attack with three duplication rounds using MNIST dataset runs 1 minute 56 second, but the same settings in Movidius result in 11 minutes and 50 seconds of run-time. We can see a huge increase in run-time by 10 minutes. In case of GTSRB it is even worse and run-time increases from 9 minutes 36 seconds to 49 minutes with the same scenario (40 minutes difference). This puts some constraints to the number of experiments we conducted using NCS. We selected to evaluate only best approaches on NCS assuming that the attack is scalable and does not depend on where the model is deployed.

Table 8: Attack results targeting Movidius neural compute stick

Dataset	**F-agreement**	**Transferability rate**
MNIST	$75.5 \pm 1.5 \rightarrow 81.7 \pm 2.2$	$33.2 \pm 3.0 \rightarrow 51.8 \pm 5.4$
GTSRB	$45.6 \pm 3.0 \rightarrow 65.9 \pm 1.7$	$36.4 \pm 1.5 \rightarrow 35.3 \pm 0.8$

Table 8 shows performance of two selected attacks on GTSRB and MNIST dataset in F-agreement and transferability rate.

In this scenario, we used slightly different models (see Section 5.3) resulting in better performance for GTSRB in both metrics and substantially worse performance in MNIST with regards to F-agreement metric. However, we can see slightly better results in the first duplication round for transferability rate in MNIST using Movidius. However, transferability rate in Movidius does not go as high as in the previous setup and stops at 52%. The reason may be that target models for Movidius are wider and have more parameters, however same number of layers. As an anecdotal evidence, the experiments on model complexity are shown in [21] and demonstrate that wider networks show different performance in F-agreement or transferability rate.

Overall, we can see that numbers are almost the same as in Section 6.2 showing that DNN extraction attacks *do not depend on where a target model deployed, but rather just on access to prediction API.*

6.4 Challenges

The main challenges that we met in this work are computational power, run-time and memory constraints put by data origin, model complexity or storing a great amount

of experimental data. To overcome such challenge we limited ourselves throughout the experiments to particular datasets, model depth and amount of queries.

C1 Computational time and power for real datasets. Throughout this work we tried to evaluate the attack on realistic models and datasets. However, complex models and datasets increase the run-time and memory requirements of the attack beyond capabilities of commodity PCs.

C2 Deployment of the models in Pytorch to Movidius device. Originally, device supports only TensorFlow and Caffe libraries [31, 20]. Many features that are used in this work are currently not supported. We implemented interface in Pytorch that allows using custom datasets DNN models on the neural stick with no support of Pytorch.

C3 Computational time is a major constraint in case of Movidius since the stick is only able to process queries one by one it may result in much slower run time than for GPU models. Using more duplication epochs have an exponential increase in run-time.

7 Related work

In this section, we describe several papers that have been recently published and closely related to ML model extraction attacks or defenses.

7.1 ML model extraction attacks

Previously in Section 2.6, we presented the two most related attacks to the work done in this thesis. Tramer et al. [41] introduce three methods to exploit confidence values returned by prediction API and extract ML model. We showed that this technique is feeble in DNN model scenarios and requires a large number of queries to achieve poor results in comparison with technique presented by Papernot et al. [36] or our techniques that we describe in this work. Papernot et al. proposed a method to extract DNN model using Jacobian-based sample generation and incremental learning of the attacker model.

Seong Joon Oh et al. [33] study neural network models from a black-box perspective and try to show the value of inner hyperparameters of a neural network. Authors list three different approaches to infer internal parameters of the target model: select a fixed set of inputs from train set and trains a new MLP classifier based on target model outputs to predict classifier parameters; craft an input to the model based on probabilities from server model to infer model's parameters one at a time; the last one is a combination of both first and second methods. In addition to inferring internal parameters paper shows that those results can be used as a boost to other attacks against the target models. These techniques can be used at the very first stage of the attack proposed in this work (Section 4) to find out model hyperparameters if it is not known to the attacker.

Daniel Lowd and Christopher Meek [28] present an attack on linear classifiers. They assume an adversary has black-box access to a prediction API and queries return only class labels. They consider SVMs and binary logistic regression classifiers as the main target. Thus, the attack works only for linear binary classifiers. Their attack uses a similar technique to Tramer et al. [41] and explores decision boundary of an attacker model by line search technique and an equation solving technique to infer weight matrices and intercept terms of a linear classifier.

7.2 Defenses against ML model extraction

Manish Kesarwani et al. [22] method introduce the first defense against model extraction attack. The technique relies on monitoring and recording all requests made by a user and computing the information gain of feature space explored by the aggregated requests. When the space explored or the information exceeds a prefixed threshold the cloud provider can raise an alarm and warn the model owner. This technique preserves model privacy and the checker just need a test dataset to estimate the information gain. Thus this technique does not provide generalized guarantees but only specific to a testing set. It may open doors for circumvention. In addition, this technique does not study request behavior and a benign user can just explore the space legitimately without performing stealing attack raising many false alarms. It can cope with a distributed attack by aggregating requests from several users together and building a single model with them. However, it does not apply to DNN models or any high-dimensional data and only limited to interpretable models, such as decision trees. Therefore, our approach cannot be detected by Manish Kesarwani et al. [22] method.

Mika Juuti et al. [21] describe first generic approach to effectively detect model extraction attacks. The idea is to analyze the distribution of queries and raise an alarm when the abnormal behaviour is noticed. The technique does not rely on whether a particular query is benign or malicious but looks at how the queries relate to each other. In addition, the technique introduced makes no assumption on the training data or the model which makes it generic. It is evaluated on the same datasets and approaches as used in this work and has a 100 % success rate of detecting all attacks presented in this thesis.

8 Conclusion

This thesis focused mainly on DNN extraction attacks. In this work, we thoroughly explored different approaches to extract DNN model in various scenarios. We presented a new query strategy to improve existing attacks [36, 41]. In this section, we draw conclusions on completed work and present some possible paths for future work.

8.1 Summary

In Section 3.2 we identified several requirements for critical evaluation of existing attacks and developing new techniques to extract a model using prediction API. To meet *Implementation requirement* we implemented two previously introduced techniques for model extraction [36, 41] using Pytorch (v0.3.0) library in Python programming language. We presented a generic approach to extract a DNN model in Section 4 and evaluation metric according to *Performance requirements*. Next, we described the data that is used in the experiments and defined several possible learning strategies that the attacker may use to boost extraction performance along with experimental setups in Section 5.3 that are used in the evaluation. We evaluate according to requirements **P1-P4** in Section 3.2. Our experiments in Section 6 show that all listed *Performance requirements* are met and *Jb-top3 dp* or *Jb-top7 p* reach up to 20 pp. performance improvement compared to previous attacks in terms of transferability rate and F-agreement (**P1** and **P2**); all *Jb-topk* techniques proposed in this work achieve higher F-agreement and transferability rate values using fewer queries than previous techniques (**P3**). Furthermore, experiments show that reliance on initial sample data using new approaches is less than Papernot et al. attack (**P4**). In addition, we evaluated the effectiveness of different learning strategies on DNN extraction attacks, e.g. dropout, use of probabilities, layer freezeing, model resetting. Dropout and probabilities help in some setups whereas model resetting and layer freezeing showed either worse or same performance and can be considered as not helpful in DNN extraction. *Scalability requirements* were satisfied by evaluating the best set-ups (*Jb-top-7 p MNIST* and *Jb-top-3 dp GTSRB*) against hardware-based ML device, e.g. Movidius Neural compute stick, and showed that it is as vulnerable as another test scenario. We showed that Movidius NCS can be used with different dataset. However, during experiments, we met several challenges that mostly relate

to computational time. It is almost infeasible to perform the attack on images resolution, such as 224×224 in CIFAR, with commodity PCs. Movidius results showed that it requires up to 10 times more time to execute the attack than ideal server setup (see Section 5.3.1). We assume that is due to the nature of Movidius API. It can only process one query at a time which is a major factor for the attack to be slower. Overall, the main goal to outperform previous approaches is achieved.

The main conclusion drawn from this work is that all tested attack methods do not depend on where the model is deployed and satisfying results can be achieved with the access to model's prediction or even better results with the use of probabilities. Therefore, results confirm a high risk of DNN model extraction from cloud-based models or even more destructive attacks on autonomous cars, where even physical access to the model can be obtained.

8.2 Future work

The results for DNN model extraction attacks show that models are vulnerable and even targeted transferability results in close to real data, such as GTSRB can be high with using bigger perturbation size for the attack. All attacks are still dependent on prior knowledge about the internal structure of the model, which can be inferred to some degree using to date methods, e.g. [33]. However, there is no work on freeing the attack from dependency on initial samples from same or similar distributions to target dataset. This would be the most valuable contribution and should be the main path for future work. Moreover, we only analyzed simplified datasets and networks due to computational power and time constraints. For future work more complex datasets and close to reality DNN networks need to be tested, e.g. ImageNet.

References

[1] Robert A Adams and John JF Fournier. *Sobolev spaces*. Vol. 140. Academic press, 2003.

[2] Charu C Aggarwal, Alexander Hinneburg, and Daniel A Keim. "On the surprising behavior of distance metrics in high dimensional space". In: *International conference on database theory*. Springer. 2001, pp. 420–434.

[3] *Amazon Machine Learning on AWS*. https://aws.amazon.com/machine-learning/.

[4] *Convolutional Neural Networks for Visual Recognition*. http://cs231n.github.io/neural-networks-1/.

[5] Dua Dheeru and Efi Karra Taniskidou. *UCI Machine Learning Repository*. 2017. URL: http://archive.ics.uci.edu/ml/datasets/.

[6] *EU General Data Protection Regulation*. https://ec.europa.eu/commission/priorities/justice-and-fundamental-rights/data-protection/2018-reform-eu-data-protection-rules_en.

[7] Matt Fredrikson, Somesh Jha, and Thomas Ristenpart. "Model inversion attacks that exploit confidence information and basic countermeasures". In: *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*. ACM. 2015, pp. 1322–1333.

[8] Feliks Ruvimovich Gantmakher and Mark Grigorevič Krejn. *Oscillation matrices and kernels and small vibrations of mechanical systems*. American Mathematical Soc., 2002.

[9] *GDPR in AI and Machine Learning*. https://www.kdnuggets.com/2018/03/gdpr-machine-learning-illegal.html.

[10] Weifeng Ge and Yizhou Yu. "Borrowing treasures from the wealthy: Deep transfer learning through selective joint fine-tuning". In: *Proc. IEEE Conference on Computer Vision and Pattern Recognition, Honolulu, HI*. Vol. 6. 2017.

[11] Ran Gilad-Bachrach et al. "Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy". In: *International Conference on Machine Learning*. 2016, pp. 201–210.

[12] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. "Deep sparse rectifier neural networks". In: *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*. 2011, pp. 315–323.

[13] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. *Explaining and Harnessing Adversarial Examples*. 2014. eprint: arXiv:1412.6572.

[14] *Google Prediction API*. https://cloud.google.com/prediction/docs/.

[15] Kaiming He et al. "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification". In: *Proceedings of the IEEE international conference on computer vision*. 2015, pp. 1026–1034.

[16] Geoffrey E Hinton et al. "Improving neural networks by preventing co-adaptation of feature detectors". In: *arXiv preprint arXiv:1207.0580* (2012).

[17] Sebastian Houben et al. "Detection of Traffic Signs in Real-World Images: The German Traffic Sign Detection Benchmark". In: *International Joint Conference on Neural Networks*. 1288. 2013.

[18] *Intel Movidius Neural Compute stick*. https://developer.movidius.com/.

[19] Roxana Istrate et al. "Incremental Training of Deep Convolutional Neural Networks". In: *arXiv preprint arXiv:1803.10232* (2018).

[20] Yangqing Jia et al. "Caffe: Convolutional Architecture for Fast Feature Embedding". In: *Proceedings of the 22Nd ACM International Conference on Multimedia*. MM '14. Orlando, Florida, USA: ACM, 2014, pp. 675–678. ISBN: 978-1-4503-3063-3. DOI: 10.1145/2647868.2654889. URL: http://doi.acm.org/10.1145/2647868.2654889.

[21] Mika Juuti et al. "PRADA: Protecting against DNN Model Stealing Attacks". In: *arXiv preprint arXiv:1805.02628* (2018).

[22] Manish Kesarwani et al. "Model Extraction Warning in MLaaS Paradigm". In: *CoRR* abs/1711.07221 (2017). eprint: 1711.07221.

[23] Murphy Kevin. *Machine learning: a probabilistic perspective*. 2012.

[24] Pavel Laskov et al. "Practical evasion of a learning-based classifier: A case study". In: *Security and Privacy (SP), 2014 IEEE Symposium on*. IEEE. 2014, pp. 197–211.

[25] Yann LeCun and Corinna Cortes. "MNIST handwritten digit database". In: (2010). URL: http://yann.lecun.com/exdb/mnist/.

[26] Zhizhong Li and Derek Hoiem. "Learning Without Forgetting". In: *Computer Vision – ECCV 2016*. Ed. by Bastian Leibe et al. Cham: Springer International Publishing, 2016, pp. 614–629. ISBN: 978-3-319-46493-0.

[27] Jian Liu et al. "Oblivious neural network predictions via minionn transformations". In: *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. ACM. 2017, pp. 619–631.

[28] Daniel Lowd and Christopher Meek. "Adversarial learning". In: *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*. ACM. 2005, pp. 641–647.

[29] *Machine-Learning-as-a-Service (MLaaS) Market - Global Forecast to 2021*. https://www.marketsandmarkets.com/Market-Reports/machine-learning-as-a-service-market-183667795.html.

[30] Aleksander Madry et al. "Towards deep learning models resistant to adversarial attacks". In: *arXiv preprint arXiv:1706.06083* (2017).

[31] Martín Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: https://www.tensorflow.org/.

[32] *Microsoft Azure Machine Learning*. https://azure.microsoft.com/en-us/services/machine-learning/.

[33] Seong Joon Oh et al. "Towards Reverse-Engineering Black-Box Neural Networks". In: *International Conference on Learning Representations*. 2018.

[34] Olga Ohrimenko et al. "Oblivious Multi-Party Machine Learning on Trusted Processors". In: *25th USENIX Security Symposium (USENIX Security 16)*. Austin, TX: USENIX Association, 2016, pp. 619–636. ISBN: 978-1-931971-32-4. URL: https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/ohrimenko.

[35] Nicolas Papernot, Patrick D. McDaniel, and Ian J. Goodfellow. "Transferability in Machine Learning: from Phenomena to Black-Box Attacks using Adversarial Samples". In: *CoRR* abs/1605.07277 (2016).

[36] Nicolas Papernot et al. "Practical Black-Box Attacks Against Machine Learning". In: *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*. ASIA CCS '17. ACM, 2017, pp. 506–519.

[37] Adam Paszke et al. “Automatic differentiation in PyTorch”. In: (2017).

[38] Dan Rahko. *The Realities of XaaS (Everything-as-a-Service)*. https://www.modernmsp.com/realities-everything-service-xaas/.

[39] Carlos Rozas. “Intel® Software Guard Extensions (Intel® SGX)”. In: (2013).

[40] Nitish Srivastava et al. “Dropout: A Simple Way to Prevent Neural Networks from Overfitting”. In: *Journal of Machine Learning Research* 15 (2014), pp. 1929–1958. URL: http://jmlr.org/papers/v15/srivastava14a.html.

[41] Florian Tramér et al. “Stealing Machine Learning Models via Prediction APIs.” In: *USENIX Security Symposium*. 2016, pp. 601–618.

[42] B. Wang and N. Zhenqiang Gong. “Stealing Hyperparameters in Machine Learning”. In: *39th IEEE Symposium on Security and Privacy*. 2018, pp. 1–19.

[43] Yuanshun Yao et al. “Complexity vs. performance: empirical analysis of machine learning as a service”. In: *Proceedings of the 2017 Internet Measurement Conference*. ACM. 2017, pp. 384–397.

[44] Fan Zhang and Kui Xu. “Annotation and Classification of an Email Importance Corpus”. In: *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*. Vol. 2. 2015, pp. 651–656.